Das SPEEDY-1050-System

Aus dem Compy-Shop-Magazin

CSM 10-88 / 11-88 / 3-89 / 4-89 (ABBUC-PD 590 / 606 / 692 / 695)

Das SPEEDY-System, Teil 1 von Peter Bee

In dieser Serie möchten wir Ihnen alles Wissenswerte über Floppy-Speeder im allgemeinen, und über die SPEEDY 1050 und die Mini-SPEEDY im besonderen vermitteln. Wir werden uns dabei aber nicht auf Werbung oder Werbeaussagen beschränken. Wir denken vielmehr an eine Reihe von Artikeln die von den verschiedenen Autoren geschrieben werden sollen.
Hier mal eine kleine Übersicht über das, was Sie alles in dieser Serie erwartet:

1. Eine allgemeine Einführung in die Problematik der Floppy-Speeder
2. Die Entwicklungsgeschichte der SPEEDY
3. Der Hardware-Ausbau der SPEEDY
4. Das Betriebssystem der SPEEDY
5. Die Sprungtabelle der SPEEDY
6. Möglichkeiten der Programmierung
7. Programm-Beispiele
8. u.s.w.

Wir werden uns allerdings nicht unbedingt an diese Reihenfolge halten! Und mal sehen, wenn Ihr Interesse groß genug ist, können wir vielleicht noch ausführlicher auf die einzelnen Themen eingehen und die Serie fortsetzen.

DIE SPEEDY 1050

Die Entwicklungsgeschichte der SPEEDY reicht bis in das Jahr 1985 zurück. Damals gab es für die Atari 1050 nur das Happy-Board als Floppy-Speeder. Aber fangen wir am Anfang an. Was ist überhaupt ein Floppy-Speeder, und was macht er?
Als Atari die ersten Computer in Deutschland verkaufte, wurde zu diesen Computern das Laufwerk mit der Bezeichnung 810 angeboten. Dieses Laufwerk war sehr robust in der Mechanik, äußerst präzise in der Anwendung. Gab es in der ersten Zeit einige Schwierigkeiten durch Datenverlust, wurde dieser Fehler durch Einbau eines Datenseparators sehr schnell beseitigt. Seit dieser Zeit gilt das Wort Datenverlust bei Atari 810 Besitzern als Fremdwort.
So gut dieses Laufwerk auch war, hatte es doch einen entscheidenden Fehler: Man konnte nur 88 kByte auf einer Diskette speichern. Dieses Single-Density Format wurde von Atari bei den 810er Laufwerken eingeführt.
Es gab auch, laut Atari Katalog, ein Laufwerk mit der Bezeichnung 815. Dieses Laufwerk wurde jedoch nie in Serie produziert. Es bestand aus zwei 810er Laufwerken die übereinander angeordnet waren. Außerdem verfügten diese Laufwerke über eine verbesserte Elektronik, die Double Density, also 176 kByte pro Diskettenseite, zuließ. Aber leider sind diese Laufwerke nie ausgeliefert worden.
Eine Verbesserung der Speicherkapazität versprach sich die Atari Welt von der Einführung des neuen Atari 1050 Laufwerks. Aber obwohl die Double Density (176 kByte) von Atari selber bei den 810er Laufwerken eingeführt wurde, "vergaß" Atari diese Speicherdichte bei den 1050er Laufwerken wieder und schraubte die Speicherkapazität auf 127 kByte herunter. Dieses etwas seltsame, Atari eigene Format bezeichnet man als Dual- oder Medium-Density.
Man kann also sagen, Atari hat hier den Anschluß an die Marktentwicklung verschlafen! Denn zu diesem Zeitpunkt gab es in den USA mehrere Fremdhersteller von Disketten-Laufwerken, die die Double-Density bereits seit Jahren nutzten. Als Beispiele seien hier nur die Laufwerke von RANA und TRAK, und die hervorragenden INDUS Laufwerke erwähnt, die teilweise sogar von Hause aus mit einem Floppy-Speeder ausgerüstet waren. Auch in Deutschland gab es zu diesem Zeitpunkt eine Firma, die doppelseitige BASF Laufwerke für den Anschluß an den Atari 800 lieferte. Mit diesen Laufwerken hatte man damals bereits das, was Atari heute mit der XF-551 liefert.

Aber die Laufwerke der Serien 810 und 1050 haben noch eine zweite große Schwäche: Die Geschwindigkeit. Zwar sind die Atari Laufwerke nicht gerade die langsamsten, aber man würde sich doch eine etwas höhere Arbeitsgeschwindigkeit wünschen. Auch hier waren die Laufwerke der Fremdhersteller wesentlich leistungsfähiger. Die 810 und die 1050 verfügen nur über 256 Byte RAM, also gerade genug um einen Sektor zwischenzuspeichern und um den Prozessor Platz zum arbeiten zu geben. Manche Fremdlaufwerke verfügten aber schon über 2 kByte und mehr! Dadurch ließ sich auch die Arbeitsgeschwindigkeit deutlich steigern.
Aus diesen Gründen, zu kleiner Speicherplatz, zu langsame Datenübertragung, sind seit dem Erscheinen der ersten Atari Laufwerke der Serie 810 verschiedene Bemühungen unternommen worden, den Laufwerken sowohl mehr Speicherplatz, als auch eine höhere Geschwindigkeit zu geben.
Für die 810 gab es zum Beispiel Umrüstsätze, damit dieses Laufwerk mit zwei Schreib-/Leseköpfen arbeiten konnte. Man konnte durch diesen Umrüstsatz nun auch die Unterseiten der Disketten beschreiben. Abgesprochen wurde diese Unterseite dann als Laufwerk 2.
Ein weiterer Grund für den Einbau eines Hardwarezusatzes ist, daß das normale Laufwerk von einem Programmierer nicht für seine eigenen Zwecke programmiert werden kann. Auch wollten viele Anwender von ihren kopiergeschützten Disketten gerne Sicherheitskopien anfertigen. Auch das geht mit einem normalen Atari Laufwerk nicht.
Wen wundert es also, daß bereits ziemlich früh einige findige Leute nach Auswegen aus diesem Dilemma suchten.
Eine der ersten Firmen, die eine gute Lösung für diese Probleme anbot, war die amerikanischen Firma Happy-Computer Inc. aus Kalifornien.
Mit Hilfe ihrer Happy 810 wurde die Arbeitsgeschwindigkeit der 810 verdreifacht. Mit diesem Board wurde auch erstmals ein Trackbuffer eingeführt. Dieser hatte die Größe von 4 kByte, konnte aber vom Anwender nicht bzw. nur sehr umständlich programmiert werden, da alle Unterlagen, insbesondere Programmbeispiele, hierfür fehlten. Es gab auch des öfteren Probleme mit dem Trackbuffer, der ab und zu Daten einfach "verschluckte". Doch die Steigerung der Arbeitsgeschwindigkeit war enorm und man konnte nun von kopiergeschützter Software Kopien anfertigen.
Es gab dann noch einige andere Floppy-Speeder, aber keiner erreichte den Bekanntheitsgrad der Happy. Als da waren zum Beispiel:

Aus Amerika:	Archiver 810
Aus Deutschland:	Clone a Disc

Die Happy 810 war ein großer Erfolg für Happy Computer Inc. Wen wunderts also, daß, etwa ein Jahr nach dem Erscheinen der ersten Atari 1050 Laufwerke, Happy Computer Inc. die ersten Happy 1050 auslieferte. Nun wurde durch die Happy 1050 nicht nur die Arbeitsgeschwindigkeit verdreifacht, sondern auch der Speicherplatz von 127 kByte auf 180 kByte erhöht. Das Problem mit dem Trackbuffer blieb den Happy 1050 Besitzern aber erhalten. Besonders bei manchen DOS-Arten.
Auch hier gab es innerhalb kurzer Zeit mehrere Produkte, die aber wiederum nicht an den Bekanntheitsgrad oder die Leistungsfähigkeit der Happy heranreichte. Beispiele:

US - Doubler	von ICD
Archiver 1050	von ICD (reine Software, funktioniert nur in Zusammenarbeit mit der Happy 1050!)
The Duplicator	von Duplicator Technologie Inc.

Zu diesem Zeitpunkt (1985) haben auch wir uns das erste Mal mit diesem Thema beschäftigt. Natürlich kannten wir die Happy 810 und die Happy 1050, hatten wir sie doch in unseren Laufwerken. Wir kannten aber auch die Probleme und waren mit den amerikanischen Produkten sehr unzufrieden. Zu hoher Datenverlust durch den Trackbuffer, zu langsame Arbeitsgeschwindigkeit, diverse Fehler im Betriebssystem, es gab keine Dokumentation für und über die Happy und somit sahen wir auch keine Möglichkeit, größere Programme für diesen Speeder zu schreiben. Kleinere Programme hatten wir ja schon eine ganze Reihe geschrieben.

Also versuchten wir mit dem Hersteller der Happy, Mr. Adams, in Verbindung zu treten und die entsprechenden Unterlagen, bzw. die entsprechende Unterstützung von ihm zu bekommen. Aber leider scheiterten wir an der Angst des Herrn Adams vor Nachbauten. Und so unberechtigt war diese Angst ja auch nicht, sieht man sich den deutschen Markt einmal genauer an. Es gibt in Deutschland ja mehr Kopien von der Happy als Originale. Das Verhältnis dürfte so bei 1:1000 liegen. So entstand bei uns der Plan, einen eigenen Floppy-Speeder zu entwickeln, der die Unzulänglichkeiten der Happy nicht haben sollte.
Die Forderungen für die Entwicklung waren von Anfang an klar:

1. Die Arbeitsgeschwindigkeit sollte größer als bei anderen sein
2. Die Datensicherheit sollte sehr groß sein
3. Erweiterbar sollte der Speeder sein
4. Das Betriebssystem sollte dokumentiert werden
5. Volle Programmierbarkeit sollte gewährleistet sein
6. Dem Benutzer sollte genügend RAM für eigene Programme zur Verfügung stehen
7. Bei zukünftigen Software-Versionen sollten auch alte Programme noch laufen

Wir wollten ein für den Anwender offenes System schaffen. Ein System mit dem jeder, der damit arbeiten wollte, dieses auch können sollte, ohne ständig Angst vor Datenverlusten haben zu müssen.
So begannen wir also mit der Entwicklung. Am Anfang benutzen wir ein umgebautes Happy-Board um unser Betriebssystem zu testen. Schnell merkten wir aber, daß wir damit nicht mehr auskamen. Unsere Software war für höhere Arbeitsgeschwindigkeiten ausgelegt als die der Happy. So hatten wir ja bereits von Anfang an einen 65SC02 als CPU für die SPEEDY vorgesehen (warum und wieso erfahren Sie in einem der nächsten Artikel).
Die Happy verfügte aber nur über einen 6502. Das zweite Problem war der Trackbuffer. Unsere Software wurde für 8 kByte ausgelegt, die Happy hatte nur 6 kByte. Schließlich, nach 4 Monaten, stand der erste Probeaufbau unserer eigenen Hardware und wurde in eine 1050 eingebaut. Zwei weitere Monate Fehlersuche (debuggen) in der Software und auf dem Board, A- und B-Testläufe in den verschiedensten Laufwerken, unter den verschiedensten Bedingungen, und wir hatten es geschafft! Das hört sich jetzt leicht und einfach an, aber wir wollten so manches mal das Handtuch werfen! Denn es gab beträchtliche Probleme mit der Arbeitsgeschwindigkeit.
Am Anfang war sie von uns höher ausgelegt worden als sie jetzt ist, aber es gab zu viele Laufwerke die mit dieser hohen Datenübertragung nicht klar kamen. So mußten wir an einigen Punkten zurückstecken. Nur bei einem nicht: die Datensicherheit!
Nach einer alles in allem 8 monatigen Entwicklungszeit, wurden die ersten SPEEDY 1050N von uns ausgeliefert. Das war am 1. Juli 1986 Gleichzeitig erhielten die bekanntesten Fachzeitungen je eine Version zum testen.
Seit diesem Zeitpunkt ist das SPEEDY-System ständig erweitert und verbessert worden. Die Systemsoftware der SPEEDY erlebte bereits 5 Updates. Dank der modularen Bauweise können Update-Versionen innerhalb von wenigen Sekunden ausgetauscht werden und stellen somit kein Problem dar.
Die Spitze dieser Entwicklungen liegt nun mit der Mini-SPEEDY vor. Hier wurde die Datensicherheit gegenüber der SPEEDY noch einmal erhöht. Auch der neue Cache-Speicher, in dem die 4 Bootsektoren und die Directory ständig im Speedy-RAM gehalten werden, hat noch mal eine Geschwindigkeitssteigerung und eine Erhöhung der Datensicherheit gebracht. Aber auch das Betriebssystem der Mini-SPEEDY läßt sich ohne Probleme in die SPEEDY 1050 übernehmen, so daß auch die Besitzer der SPEEDY diese Vorzüge nutzen können.
Heute benutzen Programmierer, Fachzeitungen, Computerclubs und Softwarehersteller SPEEDY Laufwerke zum Erstellen ihrer Disketten oder zum Schützen ihrer Software. Sehr viele geben sich dabei mit einer SPEEDY nicht zufrieden, 5 oder gar 6 Laufwerke mit SPEEDY sind die Regel. Manche besitzen auch schon Spezialversionen der SPEEDY, die nicht jedermann zugänglich sind und den Preis einer SPEEDY leicht auf 900,00 DM steigern.

Die SPEEDY ist dabei so leistungsfähig, daß mit ihrer Hilfe ein Kopierschutz erstellt werden kann, den selbst industrielle Kopierwerke nicht erzeugen können!
Ein Ende der Entwicklungen für das SPEEDY-System ist derzeit noch nicht abzusehen. Viele Programmierer nutzen heute auch die Möglichkeit, die wir ihnen mit der Programmierung der SPEEDY gegeben haben und schreiben eigene Programme. Zum Beispiel arbeitet einer dieser Programmierer an einer Parallelversion der SPEEDY, die in der Lage sein soll, 180 kByte Daten in 13 Sekunden in den Computer zu lesen. Natürlich werden solche Programmierer von uns mit Rat und Tat unterstützt.
Das war der erste Teil unserer Serie über die SPEEDY. Ich hoffe, der kleine Ausflug in die Vergangenheit und die Geschichte der SPEEDY haben Sie nicht gelangweilt. Im nächsten Monat geht es dann weiter. Bis dahin viel Spaß!

Das SPEEDY-System, Teil 2 von E. Reuß

Es fing alles mit einer ATARI 810 Floppy an. Dieses damals sehr teure und ebenso langsame Laufwerk verlangte einfach nach einem Speeder. Es dauerte eine ganze Zeit, bis die ersten Entwicklungen auf dem Markt auftauchten. Es ist heute leider nicht mehr genau nachvollziehbar, welches Produkt als erstes auf den Markt kam, fest steht jedoch, daß Ähnlichkeiten untereinander nicht auszuschließen waren. So fand man einige Routinen in der Software der 'Backup Machine', dem Vorgänger der Turbo 1050, auch in der 'Replica'.
Beide Speeder nutzten lediglich ein verändertes Diskettenformat, genauer gesagt eine andere Anordnung der Sektoren um eine Geschwindigkeitssteigerung zu erreichen. Die Erhöhung der Geschwindigkeit betrug nur etwa 10%. Wichtiger jedoch war die Tatsache, daß es nun auch möglich war, Kopien von kopiergeschützter Software anzufertigen.
Bald kam die Sensation aus Amerika: Die Happy 810. Mit diesem Speeder wurde erstmals eine echte Geschwindigkeitssteigerung erreicht. Happy Computers Inc. baute 4 kByte RAM auf die etwa 800.-- DM teure Erweiterung. Hierdurch konnten die Daten eines kompletten Tracks zwischengespeichert werden. Dieser Trackbuffer erwies sich jedoch als sehr problematisch, da die Elektronik keinen Diskettenwechsel feststellen konnte. Es passierte deshalb sehr oft, daß man das Directory auf einer anderen Diskette wiederfand.
Der hohe Preis und die schlechte Beschaffbarkeit einer originalen 'HAPPY 810' förderte die Erstellung von Raubkopien. Die HAPPY 810 bestand aus Standard-Bauteilen was den Nachbau natürlich vereinfacht hat. Es gab wohl kaum einen Besitzer einer 810, der nicht einen solchen HAPPY-Nachbau in seiner Floppy hatte. Auch ich besaß damals eine solche Erweiterung, woher auch immer. Schon bald versuchte ich die Fehler in der Software der HAPPY 810 zu beseitigen, doch zur damaligen Zeit hatte ich noch keine so große Erfahrung in der Maschinensprach-Programmierung und ich konnte das Problem mit dem Diskettenwechsel nur durch Einbau einer kleinen Zusatzschaltung beseitigen.
Die HAPPY 810 wurde zur Super-Kopierstation. Hier wurde nun endlich auch die serielle Übertragungsrate vom und zum Computer heraufgesetzt. Mit der 'WARP-SPEED' konnten Disketten für damalige Verhältnisse rasend schnell kopiert werden. Das 'WARP-SPEED-DOS', ein modifiziertes DOS 2.0, erlaubte die hohe Geschwindigkeit auch vom Basic aus zu nutzen.
Das HAPPY-Backup war das Beste überhaupt. Mit Hilfe einer kleinen Zusatzschaltung konnte die Geschwindigkeit des Antriebsmotors vom Backup gesteuert werden. Hierdurch war es möglich, Disketten zu kopieren, bei denen alle anderen Kopiersysteme versagten.
Bald kamen die ersten 1050 Laufwerke auf den Markt. ATARI hatte leider wieder an der Hardware gespart und auch in dieses Laufwerk keinen Speeder fest eingebaut. Noch viel schlimmer, statt nun endlich das echte DOUBLE-Density Format zu nutzen, wurde ein Zwischenformat eingeführt, DUAL-Density, 26 Sektoren pro Track mit 128 Bytes pro Sektor. Der Einbau eines größeren RAM hätte genügt, um das Laufwerk auch DOUBLE-Density fähig zu machen.
Auch hier waren die Entwickler gefordert, einen Speeder für dieses Laufwerk zu entwickeln. Alles wartete also auf die Entwicklung der HAPPY 1050. Die Enttäuschung war groß, als die ersten Originale in Deutschland eintrafen und sich auf der Platine ein Spezialchip befand. Dieses war ein maskenprogram-

miertes 8k ROM, das in zwei 4k Bänke unterteilt war. Ein paar schlaue Entwickler hatten schon bald die Lösung für das Problem mit dem Nachbau. 5 TTL Chips besorgten die notwendige Umschaltung und die Verwendung eines 8k EPROMs.
Schon bald erschienen die Raubkopien der HAPPY 1050 in rauhen Mengen, wußte doch keiner, wie er an ein Original aus Amerika kommen sollte. Auch ich arbeitete damals mit solch einer Raubkopie. Datensicherheit wurde bei Happy anscheinend immer sehr klein geschrieben. Auch bei der HAPPY 1050 war die Gefahr sehr groß, bei einem Diskettenwechsel Daten zu verlieren, obwohl das Laufwerk einen Wechsel sofort meldet.
Mit der HAPPY 1050 wurde es endlich möglich, echte DOUBLE-Density (180k) auf eine Diskettenseite zu bringen. Die maximale Übertragungsrate war leider bei der Original Happy 1050 genau so hoch wie das 'WARP-SPEED' der Happy 810. Auf der Happy 1050 Systemdiskette befand sich eine Datei, mit der es möglich war, die Übertragungsrate noch einmal zu erhöhen. Ultra-Speed war der neue Standard, eingeführt von ICD mit dem US Doubler.
Ein besonders pfiffiger Programmierer brachte es fertig, die Ultra-Speed Routinen fest in das Happy-EPROM einzubrennen. Nun mußte nicht jedesmal die Systemdiskette eingeladen werden, um die hohe Datenübertragung der Ultra Speed nutzen zu können. Doch das war nicht alles. Eine Trackanzeige war von HAPPY Computer Inc. ebenfalls nicht vorgesehen gewesen. Auch die hierzu notwendigen Routinen wurden nachträglich in die Original-Software integriert. Beide Erweiterungen, Ultra-Speed und Trackanzeige, stammten nicht von HCI.
Kopieren konnte jeder die Hard- und Software der Happy 1050, kaum einer brachte es jedoch fertig, die Fehler in der Software zu beseitigen. HCI hatte die Software im EPROM der Happy leider so kompliziert geschrieben, daß kaum einer durch die zum Teil sehr umständlichen Routinen durchgeblickt hat. Unnützerweise hatte HCI das 8k EPROM noch durch Bankswitching in zwei 4k Bereiche aufgeteilt, obwohl der 6502 Prozessor auf der Platine 64 kByte adressieren kann.
Hier war der Punkt gekommen, an dem wir uns ernsthaft Gedanken zu einem eigenen Floppyspeeder machten. Um die Software zu diesem Projekt zu entwickeln, mußte erst einmal die Hardware stehen. Um die Hardware zu entwickeln, mußten erst einmal die Teile der Software existieren. Unser erster Probeaufbau befand sich bald auf einer umgebauten Happy-Platine. Einige Änderungen waren notwendig um unser Konzept zu realisieren. Der 6502 Prozessor der Happy 1050 wurde durch den um mehrere Befehle erweiterten C-MOS Prozessor 65C02 ersetzt. Das aufwendige Bankswitching wurde vollständig eingespart. Für die Grundversion der SPEEDY 1050 wurden also nur noch 5 IC's benötigt. Die einzige Ähnlichkeit zur Happy 1050 bestand zum Schluß nur noch in der Adressierung der Bausteine, hier konnte keine Änderung vorgenommen werden, ein 6502-System verlangt nun mal nach einem ROM im oberen Adressbereich.
Die Grundversion der SPEEDY 1050 besteht somit aus dem Prozessor 65C02, dem EPROM 2764, dem RAM 4264, einem Gatterbaustein 74LS00 zur Erzeugung des RAM-Read/Write Signals und dem 74LS139 zur Adressdekodierung. Die Speedy-Platinen waren ursprünglich nur für 8 kByte EPROM vorgesehen. Da in der Hardware aber ein 16k-Bereich für ROM vorgesehen ist, konnte die normale SPEEDY durch das Legen einer Drahtbrücke schnell für das SPEEDY-S oder -D System vorbereitet werden.
In der Hardware der SPEEDY 1050 wurde direkt die Trackanzeige integriert. Der Adressdekoder 74LS139 stellt die hierzu benötigten Selektsignale zur Verfügung. Der Summer, oder besser gesagt Pieper wurde zum Schluß eingebaut. Die Idee hierzu kam uns, nachdem wir mehrfach die Diskette zu früh aus dem Laufwerk genommen hatten. Ein akustisches Signal sollte dem Anwender anzeigen, daß er die Diskette schnell wieder in das Laufwerk stecken soll.
Alle Bauteile, die für den Betrieb der Trackanzeige und des Miniaturlautsprechers notwendig sind, kamen mit auf die SPEEDY Grundplatine. Das Display wurde mit einem Flachbandkabel und einem Steckverbinder mit der Grundplatine verbunden.
Die Mini-Speedy ist im Prinzip eine normale Speedy 1050 ohne die Bauteile, die für die Trackanzeige und dem Miniaturlautsprecher notwendig sind. Ein paar Änderungen wurden jedoch zur normalen Speedy vorgenommen. 16k-Eproms können direkt eingesetzt werden. Zwei weiter IC's sind für die Anfang '89 vorgesehene 32k-Version vorgesehen.

Dieses war nun unsere Einführung in die Hardware der Speedy. Für Sie als zukünftige Programmierer der Speedy ist nun noch die Aufteilung der Speicherbereiche wichtig. Die Programmierung der einzelnen Bausteine wird in den folgenden Teilen unserer Serie über die SPEEDY 1050 erklärt.

Adresse:			**belegt mit:**
0000	-	0080	RAM
0100	-	0180	RAM (0000-0080 gespiegelt)
0280	-	029F	PIO 6532
0400	-	0403	Controller WD2793
		4000	Trackdisplay 1er
		4001	Trackdisplay 10er
		4002	Trackdisplay Density
		4003	Miniaturlautsprecher
8000	-	9FFF	RAM 8 kByte
C000	-	FFFF	EPROM 16 kByte
oder			
E000	-	FFFF	EPROM 8 kByte
Zusätzliche Bereiche bei der MINI-Speedy 32k-Version:			
		1000	
+		2000	Umschaltung der 16k-Bänke
C000	-	FFFF	2 mal 16 kByte ROM

Das SPEEDY-System, Teil 3 von Erwin Reuß und Peter Bee

Wenn Sie schon Besitzer einer SPEEDY sind, werden Sie die Vorteile kennen, die Ihnen dieser Hardwarezusatz für Ihr Laufwerk bietet. Vielleicht wollen Sie jetzt etwas mehr über die Möglichkeiten der Programmierung der SPEEDY erfahren.
Und genau dafür haben wir diese Serie geschrieben. Sie sollen die Möglichkeit bekommen, die speziellen Fähigkeiten der SPEEDY für Ihre eigenen Programme zu nutzen. So können Sie zum Beispiel sehr leicht die SUPER-SPEED-Routine der SPEEDY in Ihre eigenen Programme einbauen, oder eine Diskette formatieren, ohne ein DOS benutzen zu müssen, oder Sie können sich selber ein Kopierprogramm schreiben, mit dem Sie Ihre Originaldisketten kopieren können.
Sie werden dazu in dieser Serie die genaue Dokumentation des Betriebssystems, sowie eine ausführliche Dokumentation der Einsprungadressen und natürlich einige Demoprogramme finden.
Wir wenden uns mit dieser Serie an die Programmierer, die wissen, wie sie einen Floppy-Controller vom Typ 2797 oder 2793 programmieren müssen. Es würde zu weit führen, die Programmierung dieses Floppy-Controllers in dieser Serie zu erklären. Wir möchten in diesem Zusammenhang auch auf die ausgezeichnete Dokumentation des Herstellers dieses Bausteins hinweisen, die Sie in jedem guten Zubehörhandel bekommen können.
Hier also noch einmal: Diese Serie ist nicht gedacht für den Anfänger in der Maschinensprach-Programmierung! Sie ist speziell geschrieben worden für den fortgeschrittenen Programmierer.
Wichtig für einen Programmierer ist auch der folgende Hinweis: Sie brauchen an uns keine Lizenzgebühr zu zahlen, wenn Sie Software für die Speedy schreiben oder Routinen aus der SPEEDY 1050 innerhalb Ihrer Software für die SPEEDY 1050 benutzen. Sie können also Ihre Programme kommerziell vermarkten. Wir möchten Sie nur bitten, uns eine Kopie Ihres Programms zukommen zu lassen.
Und nun viel Spaß bei der Programmierung Ihrer SPEEDY 1050.

DER AUFBAU DER SPEEDY 1050 PLATINE

Es gibt drei verschiedene Versionen der SPEEDY 1050. Die erste Version ist die Grundausführung. Die zweite Version der SPEEDY 1050 ist eine erweiterte Ausführung mit Trackanzeige und einem akustischen Fehlermelder, dem Summer. Technisch, und auf die Laufwerksleistungen bezogen, sind beide Versionen identisch.
Als dritte Version ist noch die MINI-SPEEDY zu erwähnen. Diese Version ist natürlich voll Software-kompatibel zur normalen SPEEDY 1050. Die kleinere Platine läßt lediglich nicht zu, daß die Trackanzeige aufgebaut werden kann. Wenn wir also ganz allgemein von der SPEEDY reden, sind sowohl SPEEDY 1050 als auch Mini-SPEEDY gemeint.

DER AUFBAU DER GRUNDVERSION

Die Grundversion besteht aus der Platine, einem 8k-RAM-IC, einem 8k-EPROM mit dem Betriebssystem, dem Mikroprozessor 65C02 oder 65SC02 und diversen Kodier-ICs.

DER AUFBAU DER ERWEITERTEN VERSION

Zusätzlich zu den Bauteilen der Grundversion kommen bei den erweiterten Versionen der SPEEDY 1050 T Serie noch die Bauteile für den Summer und die Trackanzeige hinzu. Die Grundversion der SPEEDY 1050 läßt sich leicht durch einen entsprechenden Bausatz mit Trackanzeige und Summer nachrüsten. Nicht so die Mini-SPEEDY.

DIE FUNKTIONSWEISE DER SPEEDY

Ein normales ATARI 1050-Laufwerk besitzt einen RAM-Buffer von 256 Bytes Größe. Diesen RAM-Bereich müssen sich Datenspeicher und Mikroprozessor teilen. Da für einen Sektor ja bereits 128 Bytes gebraucht werden, hat der Mikroprozessor nicht mehr viel Platz zum arbeiten. Aus diesem Grund kann bei einem ATARI 1050-Laufwerk pro Umdrehung jeweils nur ein Sektor eingelesen und zum Computer gesendet werden. In einem Track liegen bei SINGLE DENSITY 18 Sektoren. Die Diskette rotiert mit ca. 5 Umdrehungen pro Sekunde (288 Umdrehungen/Minute).
Das ergibt für einen kompletten Track eine Ladezeit von ca. 3,6 Sekunden. Das ist die Zeit, die das ATARI 1050-Laufwerk benötigt, um einen kompletten Track einzulesen.
Die SPEEDY besitzt einen 8 kByte großen RAM-Speicher. Dieser arbeitet als Datenspooler. Pro Umdrehung der Diskette kann nun ein kompletter Track in den RAM-Buffer eingelesen werden. Das ergibt eine Ladezeit für einen Track von 0,2 Sekunden. Die Geschwindigkeit ist also um den Faktor 18 erhöht worden. Diese Geschwindigkeit ist die normale Arbeitsgeschwindigkeit eines SPEEDY Laufwerkes.
Durch die Zwischenspeicherung der Daten kann nun auch die Übertragungsgeschwindigkeit zum Computer erhöht werden. Durch diesen großen RAM-Buffer können nun auch mehr als 128 Bytes in einen Sektor geschrieben werden. Es wird also echte DOUBLE DENSITY (256 Bytes pro Sektor) möglich. Auch bei dieser Speicherdichte von nun 176 kByte auf einer Diskettenseite arbeitet das SPEEDY Laufwerk mit der hohen Geschwindigkeit.
Auf der SPEEDY Platine befindet sich neben dem RAM aber auch noch ein Mikroprozessor. Dieser Mikroprozessor ist der 65C02. Gegenüber dem 6507 einer normalen ATARI 1050 bietet der 65C02 zwei große Vorteile.
Erstens kann der 65C02 bis zu 64K adressieren (der 6507 nur 8 kBytes) und zweitens besitzt der 65C02 einen erweiterten Befehlssatz mit zusätzlichen, sehr nützlichen Befehlen. Aufgrund dieser zusätzlichen Befehle konnte das Betriebssystem der SPEEDY 1050 oder Mini-SPEEDY kurz gehalten werden, und die Geschwindigkeit der Programmausführung wird durch die geschickte Ausnutzung des erweiterten Befehlssatzes gesteigert.

DIE DATENÜBERTRAGUNG ZUM COMPUTER

Bei einem normalen ATARI 1050 Laufwerk wird, wie wir schon erwähnt haben, pro Diskettenumdrehung ein einzelner Sektor eingelesen werden und dann sofort an den Computer weitergegeben.
Diese Methode ist sehr zeitraubend. Bei einem SPEEDY Laufwerk wird bei einer Diskettenumdrehung ein kompletter Track in den RAM-Buffer eingelesen. Dadurch

hat der Computer jederzeit Zugriff auf alle Sektoren die sich in diesem Track befinden. Durch diese Zwischenspeicherung der Daten ist im Normalmodus eine Übertragung der Daten ohne Pause (bedingt durch die Ladezeit zwischen den Sektoren) möglich.
Das bedeutet, daß im Normalmodus die Lesegeschwindigkeit im Laufwerk mit maximaler Geschwindigkeit läuft, die Datenübertragung zum Computer aber mit normaler Geschwindigkeit geschieht. Aber aufgrund der weggefallenen Pausen verkürzt sich die Ladezeit um ca. 50%.
Beim Aktivieren der speziellen SPEEDY Geschwindigkeit, der SUPER-SPEED, wird die Datenübertragung vom Laufwerk zum Computer auf das Maximum gesetzt. Bei einem normalen ATARI 1050-Laufwerk geschieht die Datenübertragung zum Laufwerk mit 19.200 Baud. Beim Aktivieren der SUPER-SPEED erhöht sich diese Baudrate auf das 4 fache.

PROGRAMMIEREN DER SPEEDY
Aufgrund der besonderen Fähigkeiten der SPEEDY haben Sie nun auch die Möglichkeit das Laufwerk individuell zu programmieren.
Die Möglichkeiten, die sich Ihnen damit eröffnen, sind fast unerschöpflich. So können Sie zum Beispiel Diskettenformate nach Ihrem eigenen Bedarf erstellen, einen eigenen Kopierschutz erzeugen oder einen fremden kopieren.
Auch die SUPER-SPEED können Sie sehr leicht für Ihre eigenen Programme nutzen, wie Sie anhand eines Demoprogrammes sehen werden.

SPEICHERAUFTEILUNG SPEEDY
Nachfolgend finden Sie ein Blockschaltbild mit der genauen Speicherbelegung der SPEEDY.

$FFFF		$03FF	
	ROM - 8 KBYTE		UNBENUTZT
$E000		$0300	
$DFFF		$02FF	
	UNBENUTZT		PORT (I/O + TIMER)
$A000		$0280	
$9FFF		$027F	
	RAM - 8 KBYTE		UNBENUTZT
$8000		$0200	
$7FFF		$01FF	
	UNBENUTZT		STACK
$0404		$0100	
$0403		$00FF	
	CONTROLLER 2793/97		ZERO PAGE
$0400		$0000	

ERKLÄRUNG ZUR SPEICHERBELEGUNG
$E000 - $FFFF - BETRIEBSSYSTEM
Hier liegt das Betriebssystem Ihrer SPEEDY 1050 oder Mini-SPEEDY. Änderungen können Sie hier nicht vornehmen.
$8000 - $9FFF - ARBEITSSPEICHER
Der 8K-RAM-Block ist in 5 Bereiche unterteilt:
$9F80 - $9FFF

Hier liegen Einsprung- und Rücksprungvektoren für die Bereitschaftsroutine des Betriebssystems. Außerdem können Sie hier Erweiterungen der RESET-Routine vornehmen.

$9F00 - $9FFF

Die normale und erweiterte Kommandotabelle und die entsprechenden Einsprünge sind hier zu finden. Über das Kommando $41 können Sie diese Tabelle beliebig verändern. Diesen Befehl haben wir Ihnen bereits im SPEEDY-Handbuch erklärt, Sie werden ihn aber auch noch einmal etwas später in dieser Serie finden.

$9E00 - $9EFF

Der EXTENDED BUFFER dient zur Zwischenspeicherung von Sektordaten bei FAST WRITE oder beim SLOW MODE diverser Laufwerksfunktionen.

$8C00 - $9DFF

In diesem Bereich liegt der Trackbuffer. Hier werden bei FAST WRITE oder FAST READ erst alle Sektordaten eines Tracks zwischengespeichert. Schalten Sie die SPEEDY mit Hilfe der Menu- Diskette (Menupunkt SLOW-MODE-CONTROL) für READ SECTOR und WRITE SECTOR in den SLOW MODUS, wird dieser Speicherbereich nicht mehr vom Betriebssystem benutzt. So können Sie auch hier eigene Routinen ablegen.

$8000 - $8BFF

Freier Speicherbereich, der dem Programmierer zur Verfügung steht, also wo Sie Ihre eigenen Programme ablegen können!

$0400 - $0403 - Hier liegen die Register des Disk-Controllers 2793/97:

$0400: Lesen = Statusregister, Schreiben = Commandregister
$0401: Lesen + Schreiben = Trackregister
$0402: Lesen + Schreiben = Sektorregister
$0403: Lesen + Schreiben = Datenregister

$0280 - $02FF - Hier befinden sich die Register des Port ICs 6532 (RIOT)

Die gebräuchlichsten Register:
$0280: Port A Datenregister
$0281: Port A Richtungsregister
$0282: Port B Datenregister
$0283: Port B Richtungsregister
$0296: Timer lesen/schreiben, Timer IRQ abschalten
$029F: Timer mit Teilerverhältnis 1:1K lesen/schreiben, Timer IRQ einschalten

$0000 - $00FF - Die Zeropage

Die Zeropage und die Page 1 überlagern sich. Das heißt, Speicherstelle $0000 entspricht der Speicherstelle $0100, Speicherstelle $0001 entspricht der Speicherstelle $0101 usw. In der Zeropage stehen dem Benutzer die Speicherstellen $0090 bis $00CF zur freien Verfügung.

Soweit der erste Teil der Beschreibung des SPEEDY Betriebssystems. Nächsten Monat beginnen wir mit den Einsprungadressen.

Das SPEEDY-System, Teil 4 von Reinhard Wilde

DIE EINSPRUNGADRESSEN

Nachdem wir Ihnen im letzten Monat die Speicherbelegung der SPEEDY erklärt haben, erfahren Sie jetzt etwas über die Einsprungadressen. Wichtig ist dabei, eines zu wissen: Wir haben am Ende des ROM-Bereiches ab $FF00 eine Einsprungtabelle eingerichtet. Wenn Sie nun eine Funktion innerhalb der SPEEDY ausführen wollen, brauchen Sie nur eine oder mehrere Adressen dieser Sprungtabelle anzuspringen.

Welche Versionen es in Zukunft auch von der SPEEDY geben wird, diese Sprungtabelle wird immer an der gleichen Stelle und in der gleichen Reihenfolge erhalten bleiben. Somit ist gewährleistet, daß alle Programme, die jetzt für die SPEEDY geschrieben werden, auch auf den zukünftigen Versionen laufen, wenn die internen Routinen über diese Tabelle angesprungen werden.

Hier die Beschreibung der einzelnen Jump-Vektoren:

RESET - $FF00 - DRIVE KALTSTART

Port A und Port B (6532) werden initialisiert, der komplette RAM-Bereich gelöscht und der Disk-Controller getestet. Sollte ein Fehler beim Testen auf-

treten, erfolgt ein Sprung nach "SYSER0", wo zweimal "BELL" ausgegeben wird, anschließend folgt ein Sprung nach "RESET2".

RESET2 - $FF03 - DRIVE WARMSTART

Die System-Variablen werden neu gesetzt, das Laufwerk in den "SINGLE-DENSITY-MODUS" gebracht, der Schreib-/Lesekopf auf Track 0 justiert und jede Controller-Tätigkeit gestoppt. Zum Rücksetzen des Laufwerkes sollte diese Routine angesprungen werden, da bei "$FF00 - RESET" alle im RAM befindlichen Daten gelöscht werden.

BEREIT - $FF06 - BEREITSCHAFTSROUTINE

Zuerst wird der "SLOW"-Schalter abgefragt und das Laufwerk gegebenenfalls in den "SLOW"-Modus geschaltet. Sollten sich noch zu schreibende Sektoren im RAM befinden, wird direkt danach "TSTCO" verzweigt, damit der Motor nicht ausgeschaltet wird, falls die Laufwerksklappe geöffnet wurde. Wurde nicht verzweigt, wird die Laufwerksklappe mit der letzten Stellung verglichen. Sollte die Klappe geöffnet worden sein, werden der Antriebsmotor und der Steppermotor ausgeschaltet und der Controller-Status nach "CONST" kopiert. Ist die Klappe geschlossen worden, wird die DENSITY neu festgestellt, das Laufwerk entsprechend eingestellt und die Sektor-Folge neu eingelesen. In der "TESTCO"-Routine wird die "COMMAND"-Leitung vom Computer abgefragt. Sollte diese "gesetzt" werden, erfolgt ein Sprung nach "RDINF", wo das Kommando vom Computer eingelesen wird. Ist die "COMMAND"-Leitung nicht gesetzt, wird der Motor-Timer heraufgezählt. Sollte dieser den Wert $9800 (16 Bit) erreichen, wird "TSTWR" aufgerufen, um alle Sektoren zu schreiben, die sich noch im RAM befinden.

MOTON - $FF09 - MOTOR ZWINGEND EINSCHALTEN

Der Antriebsmotor wird ohne Rücksicht auf die Stellung der Laufwerksklappe eingeschaltet.

TSTMON - $FF0C - MOTOR EINSCHALTEN, WENN DIE LAUFWERKSKLAPPE GESCHLOSSEN IST

Erst wird die Laufwerksklappe abgefragt. Ist diese geschlossen, wird der Antriebsmotor eingeschaltet und eine Zeitverzögerung durchgeführt, um den Motor auf Touren kommen zu lassen.

MOTOFF - $FF0F - MOTOR AUSSCHALTEN

Der Antriebsmotor wird ausgeschaltet und das entsprechende Bit in "DRSTAT" gesetzt.

SDELAY - $FF12 - MOTOR TIMER EINSTELLEN

Diese Routine wird nach einer Kommandoausführung abgearbeitet. Es wird die Zeit vorgegeben, wie lange der Motor nach einer Kommandoausführung noch eingeschaltet bleiben soll.

SDRDDP - $FF15 - DRIVE DENSITY EINSTELLEN UND ANZEIGEN

Zuerst wird nach "DENDSP" gesprungen, um die aktuelle DENSITY auf dem Display anzuzeigen. Danach wird, je nach Wert in "FORKEN", der Drive-Status, die Anzahl der Sektoren pro Track und die Anzahl der Bytes pro Sektor gesetzt.

XWAIT - $FF18 - WARTESCHLEIFE KURZ

Der Wert im X-Register gibt die Anzahl der Schleifendurchläufe an. 1 Schleifendurchlauf entspricht ca. 100 Taktzyklen (0,1 msek/100 mikrosek).

X2WAIT - $FF1B - WARTESCHLEIFE LANG

Der Wert im X-Register gibt die Anzahl der Schleifendurchläufe an. 1 Schleifendurchlauf entspricht ca. 100.000 Taktzyklen (0,1 sek/100 msek).

TRACK0 - $FF1E - TRACK 0 POSITIONIEREN

Der Disk-Controller wird gestoppt und der Schreib-/Lesekopf so lange zurückgezogen, bis der Track-0-Sensor anspricht. Danach wird eine Verzögerung zum Ausschwingen der Kopfmechanik durchgeführt.

TRADJA - $FF21 - TRACK # ANZEIGEN UND SCHREIB-/LESEKOPF POSITIONIEREN

Zuerst wird die Tracknummer angezeigt und der Controller gestoppt. Ist die Klappe geschlossen, wird der Motor eingeschaltet und die Anzahl der Doppel-Steps errechnet, die durchgeführt werden müssen, um die gewünschte Kopfposition zu erreichen. Sollte der Trackwechsel über 40 Tracks gehen, wird zweimal "BELL" ausgegeben und der Kopf auf Track 0 positioniert.
Nach erfolgreicher Schreib-/Lesekopf-Positionierung wird die Tracknummer in das Track-Register des Controllers kopiert und der Schreib-/Lesekopf-Mechanik Zeit zum Ausschwingen gegeben.

TRADJ - $FF24 - SCHREIB-/LESEKOPF POSITIONIEREN

Entspricht der $FF21-Routine, mit dem Unterschied, daß die Tracknummer nur dann angezeigt wird, wenn ein Trackwechsel stattgefunden hat.

TRVR - $FF27 - 1 STEP VORWÄRTS ODER RÜCKWÄRTS AUSFÜHREN

Im Y-Register ist die Kennung für Step vorwärts oder rückwärts (plus oder minus). Die Bitposition des Steppermotors wird entsprechend herauf- oder herabgezählt und das neue Bitmuster in Port 5 des Portbausteins geschrieben.
Anmerkung: Für 1 Trackwechsel müssen immer 2 Steps erfolgen. Das heißt aber nicht, daß ohne Bedenken 80 Tracks geschrieben oder gelesen werden können.
Die beiden Steps sind verschieden lang und instabil!

CONRES - $FF2A - DISK CONTROLLER STOPPEN

Dem Disk-Controller wird der Befehl gegeben, alle laufenden Aktionen zu stoppen. In "WREADY" wird gewartet, bis der Controller den Befehl als "ausgeführt" meldet.

CONRE2 - $FF2D - CONRES WIRD ZWEIMAL AUSGEFÜHRT

Beim Einsprung in diese Routine wird die Routine $FF2A - DISK CONTROLLER STOPPEN zweimal ausgeführt.

WREADY - $FF30 - AUF CONTROLLER "IN USE-FLAG" = 0 WARTEN

Bei dieser Routine wird darauf gewartet, bis der Controller meldet, daß der letzte Befehl abgeschlossen wurde.

RD128 - $FF33 - 128 BYTES VOM COMPUTER NACH "EXBUF" HOLEN

Der Buffer wird auf "EXBUF" (Extended Buffer) und die I/O-Länge auf 128 Bytes gesetzt.

RD256 - $FF36 - 256 BYTES VOM COMPUTER NACH "EXBUF" HOLEN

Wie bei der vorhergehenden Routine wird der Buffer auf "EXBUF", aber die I/O-Länge auf 256 Bytes gesetzt.

RDBTS - $FF39 - ANZAHL DER BYTES IM ACCU IN DEN BUFFER HOLEN (X/Y-REGISTER)

Im Accu steht die Anzahl der Bytes, im X- und Y-Register die Low- und High-Adresse des Buffers, in dem die Daten vom Computer abgelegt werden sollen.
Der Timer wird gesetzt, um zu verhindern, daß der Prozessor hängen bleiben kann. Es wird jeweils ein Byte über einen indirekten Jump-Vektor vom Computer geholt (außer bei HIGH-SPEED) und in dem Buffer abgelegt. Anschließend wird die Checksumme heraufgezählt und geprüft, ob alle Bytes des Datenblocks geholt wurden. Die Checksumme wird verglichen und die I/O-Länge neu gesetzt.

RDSFOL - $FF3C - NACH VERZÖGERUNG SEKTORFOLGE VOM AKTUELLEN TRACK LESEN

Eine Verzögerungsschleife am Anfang der Routine verhindert, daß versucht wird, die HEADER schon zu lesen, wenn die Laufwerksklappe noch nicht vollständig geschlossen ist. Danach wird die Zeit gestellt, die der Controller zur Verfügung hat, um alle HEADER zu lesen. Ist ein HEADER eingelesen, wird die Track- und Sektor-Nummer auf Gültigkeit geprüft. Befindet sich die Sektornummer des gelesenen HEADERS bereits in der Sektorliste (doppelter Sektor), wird die Sektorfolge nicht mehr weiter gelesen und das Laufwerk in den Slow-Modus geschaltet. Stimmt die Anzahl der gelesenen Sektor-HEADER nicht mit der vorgegebenen (18 oder 26 Sektoren) überein, wird das Laufwerk ebenfalls in den Slow-Modus geschaltet.

RDTRA - $FF42 - AKTUELLEN TRACK IN DAS RAM EINLESEN

Die Track-Slow-Kennung wird gesetzt. Über RDHDSP wird der Kopf positioniert und der nächste HEADER eingelesen. Anschließend wird die Position der Sektor-Nummer des HEADERS in der Sektorliste gesucht. Ab der nächsten Position werden die Sektoren dann in der richtigen Reihenfolge eingelesen. Die Statuswerte der Sektoren werden in die Statusliste eingetragen. Alle Statuswerte außer $08 (CRC-ERROR) und $20 (AM-ERROR) unterbrechen dabei das Einlesen der Sektoren. Nur wenn alle Sektoren des Tracks gelesen wurden, wird die Kennung für Track-Slow zurückgesetzt.

RDTRAV - $FF45 - WIE RDTRA, JEDOCH MIT VERIFY UND 1 RETRY

Hier wird zuerst die Track- und Sektornummer errechnet und anschließend $FF42 (RDTRA) aufgerufen. Dann wird die Statusliste auf $08 (CRC-ERROR) und $20 (AM-ERROR) getestet. Ist einer der beiden Statuswerte eingetragen, wird der entsprechende Sektor nochmal gelesen.

Da sich viele Beschreibungen auf Zero-Page Adressen beziehen, hier eine Übersicht über die vom SPEEDY-System belegten Adressen:

DLYT1	= $03	Timer LO für Motor-Timer Routine
DLYT2	= $04	Timer HI für Motor-Timer Routine
LDSW	= $05	Letzte SLOW-Schalter Position
WRKEN	= $06	Anzahl der zu schreibenden Sektoren im RAM
EXSECT	= $07	Sektornummer der Daten im extended Buffer
DUMKEN	= $08	SLOW-Status
FORKEN	= $09	Aktuelle Density $00=DD, $41=MD, $82=SD
FORKEN2	= $0A	Density für Format, wenn durch COM4F gesetzt
LWRTRA	= $0B	Tracknummer der zu schreibenden Daten im RAM
LTRACK	= $0C	Tracknummer des zuletzt gelesenen Sektors
TRACK	= $0D	Aktuelle Tracknummer
SECTOR	= $0E	Aktuelle Sektornummer
CONST	= $0F	Controller-Status
DRSTAT	= $10	Drive-Status
COMST	= $11	Command-Status
RETRY	= $12	Anzahl der Retries für Read/Write, normal = 2
RWLEN	= $13	I/O Länge
SECLEN	= $14	Anzahl der Bytes pro Sektor
USKEN	= $15	Kennung für Übertragungsgeschwindigkeit
DLYTIM	= $16	Enthält die Zeit, wie lange der Motor nach einem Befehl noch läuft
STPTIM	= $17	Verzögerung für Steppermotor
COMPOS	= $18	Position des letzten Befehls in der Kommandotabelle
IND	= $19	Indirekt-Vektor für Datenbuffer
CHKSUM	= $1B	Checksum für Datenübertragung
RDDATK	= $1C	Kennung, ob Daten vom Computer geholt werden müssen
KLAPPE	= $1D	Letzte Klappen Position
SECANZ1	= $1E	Sektoranzahl pro Track, die vorhanden sind
SECANZ	= $1F	Sektoranzahl pro Track, die vorhanden sein sollen
SECLST	= $20	Sektorenliste
STALST	= $40	Sektoren-Status-Liste
STPPOS	= $60	Bit-Position für Steppermotor

DSPCTR = $61 Display/Drive-Controlbyte

BLOCKS = $62 Anzahl der Datenblocks für Datenübertragung

COUNTL = $64 16-Bit Zähler für Datenübertragung (low-Byte)

COUNTH = $65 16-Bit Zähler für Datenübertragung (high-Byte)

RDIND = $68 Indirekter Vektor für 1 Byte vom Computer empfangen

SDIND = $6A Indirekter Vektor für 1 Byte zum Computer senden

Die Zeropageadressen $90 bis $CF sind unbenutzt und frei für eigene Programme.

TSTWR - $FF48 - NOCH ZU SCHREIBENDE SEKTOREN AUS DEM RAM AUF DIE DISKETTE SCHREIBEN

In "WRKEN" steht die Anzahl der zu schreibenden Sektoren. Ist der Wert gleich 0, wird die Routine sofort verlassen. Sonst wird der Motor eingeschaltet, die Tracknummer für die zu schreibenden Sektoren aus "LWRTRA" nach "TRACK" kopiert und der Schreib-/Lesekopf positioniert. Anschließend wird der nächste HEADER eingelesen und die Sektornummer in der Sektorliste gesucht. Ist der Sektor nicht eingetragen (zum Beispiel bei geschützten Disketten, die mit "FAST-WRITE" beschrieben wurden), wird 1 Bell ausgegeben. Ansonsten wird ab der gefundenen Sektorposition die Statusliste auf negative Werte (Kennung für zu schreibende Sektoren) überprüft. Wird ein solcher Wert gefunden, wird der entsprechende Sektor geschrieben und in der Statusliste als geschrieben ($40) eingetragen. "WRKEN" wird um 1 heruntergezählt und die Position in der Sektorliste heraufgezählt. Sollte beim Schreiben eines Sektors ein Fehler auftreten, indem der "WRITE-PROTECT-Schalter" umgeschaltet war oder die Laufwerksklappe geöffnet wurde, wird die Routine "WRERR" aufgerufen, in der dem Anwender auf optischem und akustischem Wege ca. 5 Sekunden Zeit gegeben wird, um die Laufwerksklappe wieder zu schließen. Alle "WRITE-PROTECTIERTEN" Sektoren werden in der Statusliste als geschrieben ($40) eingetragen.

TSTDAT - $FF4B - TSTWR AUFRUFEN UND STATUSLISTE MIT $40 FÜLLEN

Diese Routine muß abgearbeitet werden, wenn das Laufwerk in den "SLOW-MODE" geschaltet wurde, damit alle Sektoren geschrieben bzw. als nicht gelesen markiert werden.

SD128B - $FF4E - 128 BYTES VOM EXBUF ZUM COMPUTER SENDEN

Die Übertragungslänge wird auf 128 Bytes und der Buffer auf "EXBUF" gesetzt.

SD256B - $FF51 - 256 BYTES VOM EXBUF ZUM COMPUTER SENDEN

Die Übertragungslänge wird auf 256 Bytes und der Buffer auf "EXBUF" gesetzt.

SDBTS - $FF54 - ANZAHL DER BYTES IM ACCU AUS BUFFER (X/Y-REGISTER) SENDEN

Es wird jeweils 1 Byte aus dem Buffer geladen, die Checksumme heraufgezählt und über die Jump-Tabelle der Senderoutinen gesprungen, um das gelesene Byte zum Computer zu senden. Dieser Vorgang wird in einer Schleife solange wiederholt, bis der gesamte Buffer gesendet wurde. Anschließend wird die Checksumme gesendet.

SEND41 - $FF57 - QUITTUNG $41 ZUM COMPUTER SENDEN

Der Accu wird mit dem Wert $41 ("A") geladen, und nach einer Verzögerung (damit die Quittung an den Computer nicht zu schnell kommt) wird über die Jump-Tabelle der Senderoutinen gesprungen, um die Quittung in der richtigen Übertragungsgeschwindigkeit zu senden. (A=Acknowledge)

SEND43 - $FF5A - QUITTUNG $43 ZUM COMPUTER SENDEN

Dasselbe wie bei "SEND41"-$FF57, jedoch mit dem Quittungsbyte $43 (C=Complete).

SEND45 - $FF5D - QUITTUNG $45 ZUM COMPUTER SENDEN

Dasselbe wie bei "SEND41"-$FF57, jedoch mit dem Quittungsbyte $45 (E=Error).

SEND4E - $FF60 - QUITTUNG $4E ZUM COMPUTER SENDEN

Dasselbe wie bei "SEND41"-$FF57, jedoch mit dem Quittungsbyte $4E (N=Negative Acknowledge).

RDSECT - $FF63 - AKTUELLEN SEKTOR VON DER DISKETTE IN DEN VORBEZEICHNETEN RAMBEREICH EINLESEN.

Sektornummer in das Sektor-Register kopieren. "READ-SEKTOR"-Befehl an den Computer geben. "TIME-OUT"-Zeit setzen. Nun werden die Daten Byte für Byte vom Controller übernommen und in den bezeichneten Buffer (indirekt "IND") abgelegt. Sind alle Daten gelesen, wird auf den Controller gewartet, bis dieser seine Arbeit eingestellt hat, und der Status des gelesenen Sektors in das Status-Register des Controllers übergeben wurde. Sollte ein "TIME-OUT" aufgetreten sein, wird noch ein Leseversuch gestartet.

RDSEC1 - $FF66 - BEZEICHNETEN SEKTOR IN BEZEICHNETES RAM EINLESEN

Die gleiche Routine wie RDSECT-$FF63. Nur muß die Sektornummer bereits in das Sektor-Register des Controllers geschrieben sein.

WRSECT - $FF69 - AKTUELLEN SEKTOR VON VORBEZEICHNETER RAMADRESSE AUF DIE DISKETTE SCHREIBEN

Sektornummer in das Sektor-Register kopieren. "WRITE-SECTOR"-Befehl an den Controller geben. "TIME-OUT"-Zeit setzen. Nun werden die Daten Byte für Byte an den Controller übergeben.
Sind alle Daten übergeben, wird auf den Controller gewartet, bis dieser seine Arbeit eingestellt hat und der Write-Status vom Controller übernommen wird. Sollte ein "TIME-OUT" aufgetreten sein, wird überprüft, ob der Controller noch arbeitet. Wenn ja, wird noch ein Schreibversuch gestartet.

TSTWRP - $FF6F - "WRITE-PROTECT" UND LAUFWERKSKLAPPE TESTEN

Es wird "CONRES" aufgerufen, wo der Disk-Controller seine augenblickliche Arbeit unterbricht und der Controller-Status gelesen wird. Anschließend werden Bit 6 (WRITE-PROTECT) und Bit 7 (LAUFWERKS-KLAPPE) ausmaskiert. Ist eines der beiden Bits gesetzt, können keine Daten geschrieben werden!

VERSEC - $FF72 - AKTUELLEN SEKTOR MIT ANGEGEBENEM RAM VERGLEICHEN

Sektornummer in das Sektor-Register kopieren. "READ-SEKTOR"-Befehl an den Controller geben. "TIME-OUT"-Zeit setzen. Nun werden die Daten Byte für Byte vom Controller übernommen und mit der bezeichneten Adresse (indirekt "IND") verglichen. Ist ein Wert ungleich, wird das Lesen unterbrochen, der Controller gestoppt, die Kennung für "DATEN UNGLEICH" (ACCU<>0) gesetzt und CARRY für "KEIN LESEFEHLER AUFGETRETEN" gelöscht. Stimmen alle Daten mit dem angegebenen Buffer überein, wird die Kennung für "DATEN GLEICH" (ACCU=0) und "KEIN LESEFEHLER" (CARRY=0) gesetzt. Tritt während des Vergleichens ein "TIME-OUT" auf, wird geprüft, ob der Controller noch arbeitet. Wenn ja, wird der Vergleich der Daten fortgesetzt. Ansonsten wird "CARRY" gesetzt (KENNUNG FÜR LESEFEHLER).

VERSE1 - $FF75 - BEZEICHNETEN SEKTOR MIT ANGEGEBENEN RAM VERGLEICHEN

Die gleiche Routine wie VERSEC-$FF72. Nur muß die Sektornummer in das Sektor-Register des Controllers geschrieben sein.

STELL - $FF78 - COM-STATUS AUF ERROR UND 2 RETRIES SETZEN

"RETRY" wird auf 2 Versuche und "COMST" vorsorglich auf "COMMAND-ERROR" gesetzt.

QUITT - $FF7B - QUITTUNG "C" ODER "E" ZUM COMPUTER SENDEN

Den Controller-Status übernehmen. Wenn "CONST" auf "COMMAND-ERROR" steht, wird die Kennung für "FEHLER BEI LETZTER LAUFWERKS-OPERATION" in "DRSTAT" gesetzt. Wenn Bit 7 und Bit 0 in "DSPCTR" gesetzt sind, wird der Controller-Status auf dem Display angezeigt, 1 "BELL" ausgegeben und die Quittung $45 ("E") zum Computer gesendet. Ist "COMMAND-STATUS" o.k., wird die Kennung für "LAUFWERKS-OPERATION IN ORDNUNG" gesetzt und die Quittung $43 ("C") zum Computer gesendet.

RDHEAD - $FF7E - DIE NÄCHSTEN HEADERDATEN EINLESEN
"TIME-OUT"-Zeit setzen. "READ-HEADER"-Befehl an den Controller geben. Nun werden die 6 Bytes des nächsten auffindbaren HEADERS in einer Schleife eingelesen und ab der Adresse $7A abgelegt. In "WREADY" wird darauf gewartet, daß der Controller seine Arbeit einstellt. CARRY als Lesefehler-Flag wird zurückgesetzt.

RDHD1 - $FF81 - WIE RDHEAD, ABER TIMER NICHT SETZEN
Dasselbe wie $FF7E. Nur muß "TIME-OUT" bereits gesetzt sein.

CALCTS - $FF87 - TRACK- UND SEKTORNUMMER ERRECHNEN
Sektornummer LOW und HIGH werden zum "IND"-Pointer kopiert (für RAM/ROM-Adressen) und auf Nummer=0 oder Nummer>$7FFF geprüft. Ist das der Fall, wird in "DUMKEN" noch der SLOW-Status getestet. Andernfalls wird die Sektornummer in IND/IND+1 solange um die Anzahl der Sektoren pro Track herabgezählt, bis sie die Nummer 0 unterschreitet. Als Ergebnis hat man die gewünschte Track- und Sektornummer. Die Tracknummer wird noch mit dem Wert 40 verglichen (Tracknummer >39). Das CARRY-Flag wird durch den Vergleich entsprechend gesetzt. Nach der Rückkehr aus dieser Routine stehen die Prozessor-Status-Flags wie folgt:
C=1 SEKTORNUMMER UNZULÄSSIG
N=1 RAM/ROM-ADRESSE
Z=1 ZERO-PAGE-ADRESSE 0

SETBUF - $FF8A - BUFFER NACH AKTUELLEM SEKTOR SETZEN
Je nach Sektorlänge wird die Sektornummer durch 2 geteilt (oder nicht) und zur Anfangsadresse des Datenbuffers hinzu addiert. Die Bufferadresse befindet sich dann in IND/IND+1.

SETBU2 - $FF8D - BUFFER NACH SEKTORNUMMER IM ACCU SETZEN
Entspricht der Routine "SETBUF"-$FF8A, jedoch muß die Sektornummer (1..26) bereits im ACCU stehen.

SEXBUF - $FF90 - ADRESSE DES EXTENDED BUFFERS SETZEN
Die Adresse von "EXBUF" wird nach IND/IND+1 geladen.

SETRWL - $FF93 - ANZAHL DER BYTES FÜR DEN ZU ÜBERTRAGENDEN DATENBLOCK SETZEN
Bei Sektornummern von 1 bis 3 wird die Übertragungslänge auf 128 Bytes, ansonsten auf den Wert von "SECLEN" gesetzt. Die Anzahl der Datenblöcke wird auf 1 gesetzt.

COPSLT - $FF96 - SEKTORLISTE FÜR AKTUELLE DENSITY IN ZEROPAGE KOPIEREN
In "SDRDDP" werden die Werte für die aktuelle DENSITY richtig gesetzt und die DENSITY auf dem Display angezeigt. Anschließend wird, je nach Wert in "FORKEN" (DENSITY-Kennung), die entsprechende Sektorliste für SINGLE- oder DOUBLE-DENSITY nach "SECLST" ($20) kopiert.

BELL1 - $FF99 - 1 MAL BELL ÜBER DEN SUMMER AUSGEBEN
Der Summer wird mittels einer Verzögerungsschleife mit einer bestimmten Frequenz angesteuert.

CLRDSP - $FF9C - DISPLAY ABSCHALTEN
In die Display-Speicherstellen $4000, $4001 und $4002 werden Nullen geschrieben. Die Speicherstellen sollten nur im Schreibzugriff (z.B. STA $4000) angesprochen werden, da sonst auf dem Display unkontrollierte Zeichen erscheinen.

TRAANZ - $FF9F - AKTUELLE TRACKNUMMER ANZEIGEN
Der ACCU wird mit dem Wert von "TRACK" geladen, und es wird, je nach Wert in "DSPCTR" zur Dezimal- oder Hexadezimal-Ausgaberoutine gesprungen.

DEZOUT - $FFA2 - WERT IM ACCU WIRD IN DEZIMALER FORM ANGEZEIGT
Der Wert im ACCU wird in einer Schleife um 10 heruntergezählt, bis er den Wert 0 unterschreitet. Der Schleifenzähler entspricht dann dem Wert für das 10er-Stellen-Display und die Restsumme dem Wert für das 1er-Stellen-Display.

Die Werte für die richtige Segmentsteuerung werden einer Konstantentabelle (SEGTBL) entnommen.

HEXOUT - $FFA5 - WERT IM ACCU WIRD IN HEXADEZIMALER FORM ANGEZEIGT
Zuerst werden die unteren 4 Bits des Wertes im ACCU ausmaskiert, die dem Wert für das rechte Display entsprechen, dann die oberen 4 Bits.

DENDSP - $FFA8 - AKTUELLE DENSITY AUF DEM DISPLAY ANZEIGEN
Je nach Wert in "FORKEN" (DENSITY-Kennung) werden die entsprechenden Segmente des Displays angesteuert.

SETTIM - $FFAB - TIMER MIT DEM WERT IM ACCU SETZEN
Timer-Interrupt-Flag wird gelöscht und der Timer mit dem Wert im ACCU gestartet.

CRLTRA - $FFAE - EINEN TRACK REFORMATIEREN
In "FSTART" wird das "WRITE-TRACK"-Kommando gestartet und der Timer gesetzt. Nun wird der Track mit dem Wert $00 beschrieben, bis der Timer abgelaufen ist. Es wird der Track "gelöscht", auf dem sich der Schreib-/Lesekopf befindet.

CLRDSK - $FFB1 - GANZE DISKETTE REFORMATIEREN
Hier werden alle Tracks nacheinander, beginnend bei Track 39 (39..0) gelöscht. Der Schreib-/Lesekopf wird jeweils positioniert und "CLRTRA" aufgerufen.

RAMTST - $FFB4 - TEST DES LAUFWERKINTERNEN RAMS
Im ersten Teil wird die Zero-Page getestet. Der Wert der Speicherstelle, die getestet wird, wird jeweils in EXBUF+1 zwischengespeichert. Zuerst wird die Zero-Page mit dem Wert $55 getestet, das heißt, der Wert $55 wird in jede Speicherstelle geschrieben und wieder gelesen.
Ist der Wert gleich geblieben, ist die Speicherstelle in Ordnung. Anschließend wird die Zero-Page noch einmal mit dem Wert $AA getestet. Ist während des Tests kein Fehler festgestellt worden, wird der Speicherbereich von $8000 bis RAMTOP ($A000) auf die gleiche Art getestet wie die Zero-Page. Tritt bei einer Speicherstelle ein Fehler auf, wird die Adresse jener Speicherstelle in $90/$91 abgelegt und der Test wird abgebrochen.
Ist kein RAM-Fehler festgestellt worden, steht in $90/$91 die höchste RAM-Adresse. Nach Abschluß der RAM-Testroutine wird die Adresse, die in $90/$91 steht, zum Computer gesendet.

ROMTST - $FFB7 - ROM-TEST
Vorsorglich wird Command-Status auf ERROR gesetzt. In IND/IND+1 wird die Adresse $E000 gesetzt. Anschließend wird für eine Page die Checksumme errechnet und mit den Originalwerten in einer Tabelle verglichen. Ist die Checksumme gleich, wird die High-ROM-Adresse in $91 um 1 heraufgezählt und die nächste ROM-Page getestet. Insgesamt werden 32 ROM-Pages ($E000 bis $FFFF) getestet. Stimmen alle Checksummen mit denen der "ROMCHK"-Tabelle überein, wird der Command-Status zurückgesetzt und die Quittung ("C") zum Computer gesendet.
Ist ein Fehler gefunden worden, wird der Command-Status nicht zurückgesetzt und die Quittung ("E") zum Computer gesendet.

SPEEDT - $FFBA - MOTOR-SPEED-TEST
Vorsorglich Command-ERROR setzen und Schreib-/Lesekopf auf Track 0 positionieren. In "FDSEC1" wird Sektor 1 zweimal direkt hintereinander gelesen und die Zeit über Taktzyklen gemessen. Dann wird in einer Schleife die gemessene Zeit von der Konstanten $C0E1E4 solange abgezählt, bis der Wert 0 unterschritten wird. Die nachfolgende Nachkomma-Stellenrundung wird mit dem Rest der vorhergehenden Rechnung vorgenommen, indem die gemessene Zeit durch 2 geteilt und vom Rest der vorhergehenden Rechnung abgezogen wird. Ist das CARRY-Flag dann gesetzt, wird die Nachkommastelle um 1 erhöht. Der aus 2 Daten bestehende Speed-Wert wird zum Computer gesendet. Der hexadezimale Wert $2875 bedeutet dabei 287,5 UPM.

Erweiterte JMP-Tabelle ab Speedy V1.4

TSTDEN - $FFBD - FORMAT DER DISKETTE FESTSTELLEN
Der Schreib-/Lesekopf wird auf Track 0 positioniert und das Laufwerk im MFM-Mode gesetzt. Anschließend wird mit "RDHEAD" getestet ob sich ein Header lesen läßt. Gelingt das, wird die Routine verlassen (N-Flag=0 für Test ok.). Ansonsten wird nochmal im FM-Mode (Single-Density) versucht ein Header zu lesen. Mißlingt auch dies, wird der gesamte Vorgang noch auf den Tracks 1 bis 3 durchgeführt. Sollten auch danach noch keine Header gefunden worden sein, steht das Laufwerk in Single-Density und die Routine wird mit gesetztem N-Flag (Fehler-Flag) verlassen. Falls vorhanden, wird der Summer kurz mit einem Ton angesteuert und auf dem Display erscheint "nF" für nicht formatiert.

FSTART - $FFC0 - "WRITE-TRACK" KOMMANDO STARTEN
Der Schreib-/Lesekopf wird auf dem Track positioniert, der in "TRACK" festgelegt ist. Danach wird der "WRITE-TRACK"-Befehl gestartet und der Time-Out für eine Diskettenumdrehung festgelegt. Ein gesetztes Carry-Flag nach Abarbeitung dieser Routine bedeutet, daß ein Fehler aufgetreten ist.

FORMTR - $FFC3 - AKTUELLEN TRACK FORMATIEREN
"FSTART" wird aufgerufen und anschließend der aktuelle Track in dem Format formatiert, das in "FORKEN" festgelegt ist. ACHTUNG: der Schreib-/Lesekopf muß bereits auf dem Track positioniert sein, der formatiert werden soll. Fehlerflag ist das gesetzte Carry-Flag.

FORDSK - $FFC6 - DISKETTE FORMATIEREN
Die Diskette wird in dem Format formatiert, das in "FORKEN2" festgelegt ist. Die Tracks werden beginnend mit Track 39 bis herunter auf Track 0 formatiert. Fehlerflag ist das gesetzte Bit 6 in COMST (Speicherstelle $11).

WRBOOT - $FFC9 - BOOTSEKTOREN UND VTOC-SEKTOR(EN) SCHREIBEN
Es werden zunächst die Sektoren 1 bis 3 mit einem Bootprogramm beschrieben, in dem gemeldet wird, daß die Diskette mit der Speedy formatiert wurde. Je nach Format der Diskette wird der/die VTOC-Sektor(en) $168 ($400) beschrieben.

DISPL - $FFCC - WERT IN X/Y-REGISTER AUF DEM DISPLAY ZEIGEN
Die Werte, die sich bei Aufruf dieser Routine in den Registern X und Y befinden, werden auf dem Display (falls vorhanden) angezeigt.

SENDEN - $FFCF - DATEN ZUM COMPUTER SENDEN
Parameterübergabe: Länge des zu übertragenden Datenblocks in den Registern X und Y. In den Speicherstellen IND/IND+1 ($19/$1A) muß die Bufferadresse stehen. Die Checksumme des Datenblocks wird nicht gesendet. Dies hat den Zweck, das mehrere Datenblöcke direkt nacheinander ohne Unterbrechung gesendet werden können. Die Checksumme muß man extra senden lassen. Das Checksummenbyte steht in der Speicherstelle CHKSUM ($1B).

INSTALL - $FFD2 - INSTALLIEREN EINES NEUEN KOMMANDOS
Hiermit wird man in Zukunft neue Befehle in der Kommandotabelle installieren oder bereits existierende Kommandos wieder löschen können. Diese Routine entspricht der, die aufgerufen wird, wenn vom Computer der Befehl $41 gegeben wird. Nur können dann die Kommandos auch von Programmen innerhalb der Speedy ein- und ausgetragen werden.

Das war nun die vollständige Beschreibung aller Einsprungadressen. Einige Erklärungen beziehen sich auf Zeropage-Adressen, deren Bedeutung bereits in Teil 4 beschrieben wurde.
Damit wäre die Serie über das SPEEDY-System zunächst einmal abgeschlossen. Programmierbeispiele werden in einer der nächsten Ausgaben folgen.
Als Programmierhilfe ist die BIBO-Assembler Tooldisk 2 zu empfehlen. Auf dieser zweiseitigen Disk finden Sie das komplette dokumentierte Listing eines Sektorkopierers und eines Backup-Programmes für geschützte Software.

Das SPEEDY-System, aus dem SPEEDY-Anwender-Handbuch Version 1.0 vom 17.10.1986 (c) 1986 Compy-Shop

DIE BEFEHLE DER SPEEDY 1050
So, nachdem Sie nun mehr über die Einsprungadressen wissen, und bevor wird zum ROM-Listing kommen, hier nun nochmal alle Befehle der SPEEDY 1050 und ihre Anwendung.
KOMMANDO ist der Wert, der sich vor Aufruf der SIO - Routine ($E459) in der Speicherstelle $0302 befindet.
AUX1 und AUX2 entsprechen den Werten, die sich in den Speicherstellen $030A und $030B befinden. Bei einigen Befehlen werden AUX1 und AUX2 nicht benutzt und dürfen beliebige Werte annehmen.
Die Befehle sind im übrigen nicht nach den Hexadezimalnummern geordnet, sondern nach ihrer Funktion:

KOMMANDO: $52
FUNKTION: Sektoren lesen
AUX1: Sektornummer oder ROM-Adresse LOW BYTE
AUX2: Sektornummer oder ROM-Adresse HIGH BYTE
BESCHREIBUNG: Es werden je nach DENSITY 128 oder 256 Bytes gesendet. Sektoren 1-3 sind immer 128 Bytes lang.

KOMMANDO: $50
FUNKTION: Sektoren schreiben ohne Verify
AUX1: Sektornummer oder ROM-Adresse LOW BYTE
AUX2: Sektornummer oder ROM-Adresse HIGH BYTE
BESCHREIBUNG: Das Laufwerk erwartet je nach DENSITY 128 oder 256 Bytes. Sektoren 1-3 sind immer 128 Bytes lang.

KOMMANDO: $57
FUNKTION: Sektoren schreiben mit Verify
AUX1: Sektornummer oder ROM-Adresse LOW BYTE
AUX2: Sektornummer oder ROM-Adresse HIGH BYTE
BESCHREIBUNG: Das Laufwerk erwartet je nach DENSITY 128 oder 256 Bytes. Sektoren 1-3 sind immer 128 Bytes lang.

KOMMANDO: $53

FUNKTION: Laufwerkstatus

AUX1: nicht benutzt

AUX2: nicht benutzt

BESCHREIBUNG: Das Laufwerk sendet 4 Bytes (zur Adresse $02EA-$02ED), die den Status der letzten Diskettenoperation beinhalten.

Byte 1: Drive Status
- Bit 0 - COMMAND FRAME ERROR
- Bit 1 - CHECKSUM ERROR
- Bit 2 - OPERATION ERROR
- Bit 3 - WRITE PROTECT
- Bit 4 - MOTOR ON
- Bit 5 - DOUBLE DENSITY
- Bit 6 - unbenutzt
- Bit 7 - DUAL DENSITY

Byte 2: Controller Status
- Bit 0 - BUSY
- Bit 1 - DRQ
- Bit 2 - LOST DATA
- Bit 3 - CRC ERROR
- Bit 4 - RECORD NOT FOUND
- Bit 5 - RECORD TYPE
- Bit 6 - WRITE PROTECT
- Bit 7 - NOT READY

Byte 3: Time-Out Wert für Format Disk ($E0)

Byte 4: unbenutzt (immer 0)

KOMMANDO: $21

FUNKTION: Formatiere Diskette (SINGLE/DOUBLE DENSITY)

AUX1: nicht benutzt

AUX2: nicht benutzt

BESCHREIBUNG: Dieses Kommando wird benutzt um Disketten in SINGLE- oder DOUBLE-DENSITY (720 Sektoren) zu formatieren. Das DENSITY-Format wird durch einen vorherigen $4F-Befehl (Laufwerkskonfiguration) eingestellt. Wird das Laufwerk nach dem Einschalten nicht konfiguriert, wird automatisch in SINGLE-DENSITY formatiert. Das Laufwerk sendet nach dem Formatieren je nach DENSITY 128 oder 256 Bytes an den Computer. Die ersten zwei Bytes müssen immer $FF sein.

KOMMANDO: $22

FUNKTION: Formatiere Diskette (MEDIUM DENSITY)

AUX1: nicht benutzt

AUX2: nicht benutzt

BESCHREIBUNG: Dieses Kommando wird benutzt um Disketten in DUAL-DENSITY (MEDIUM = 1040 Sektoren) zu formatieren. Es werden immer 128 Bytes zum Computer gesendet. Die ersten beiden Bytes müssen immer $FF sein.

KOMMANDO: $20

FUNKTION: Automatisches Formatieren

AUX1: Konfigurationsbyte

AUX2: nicht benutzt

BESCHREIBUNG: Dem Laufwerk wird nur der Befehl zum Formatieren gegeben. Es wird sofort ein 'Complete' zurückgesendet. Mit diesem Befehl können alle drei Formate, abhängig vom Konfigurationsbyte ($00=SINGLE, $20=DOUBLE, $80=MEDIUM) generiert werden. Ein Write-Protect wird sofort zurückgemeldet.

Fehler beim Formatieren können dem Computer nicht gemeldet werden, da keine Daten nach Befehlsausführung zurückgesendet werden. Der Formatierungsvorgang und eventuelle Formatierungsfehler können auf dem Display verfolgt werden. Abhängig vom Drive/Display-Status Befehl wird nach dem Formatieren automatisch die VTOC (DOS 2.5 kompatibel) und 3 Bootsektoren geschrieben.

KOMMANDO: $3F

FUNKTION: SIO-Geschwindigkeitsbyte ermitteln

AUX1: nicht benutzt

AUX2: nicht benutzt

BESCHREIBUNG: Es wird ein Byte zum Computer gesendet, das die HIGH SPEED Übertragungsgeschwindigkeit beinhaltet. Dieses Byte wird für die HIGH SPEED SIO-Routine benötigt und beträgt bei der SPEEDY 1050 normalerweise $09.

KOMMANDO: $4E

FUNKTION: Laufwerkskonfiguration auslesen

AUX1: nicht benutzt

AUX2: nicht benutzt

BESCHREIBUNG: Es werden 12 Bytes der Konfigurationstabelle zum Computer gesendet. Die Bedeutung der einzelnen Bytes sind:
Byte 1 - Anzahl der Tracks (immer 40)
Byte 2 - Step Rate (immer 1)
Byte 3 - Sektoren pro Track HIGH (immer 0)
Byte 4 - Sektoren pro Track LOW (18 oder 26)
Byte 5 - Anzahl der Köpfe (immer 0)
Byte 6 - Aufzeichnungsformat (0=FM/4=MFM)
Byte 7 - Bytes pro Sektor HIGH (1=256/0=128)
Byte 8 - Bytes pro Sektor LOW (0=256/128=128)
Byte 9 - Laufwerk aktiv (immer 255)
Byte 10 - unbenutzt (immer 0)
Byte 11 - unbenutzt (immer 0)
Byte 12 - unbenutzt (immer 0)

KOMMANDO: $4F

FUNKTION: Laufwerk konfigurieren

AUX1: nicht benutzt

AUX2: nicht benutzt

BESCHREIBUNG: Dieser Befehl wird benutzt um das Laufwerk für den nächsten Formatierungsbefehl einzustellen. Das Laufwerk erwartet 12 Bytes, die genau der Reihenfolge der vorherigen Bytes ($4E) entsprechen müssen.

KOMMANDO: $51

FUNKTION: Schreibvorgang beenden

AUX1: nicht benutzt

AUX2: nicht benutzt

BESCHREIBUNG: Nach jedem Schreibbefehl wartet das Laufwerk ca. 2 Sekunden bis die Daten aus dem Trackbuffer auf die Diskette geschrieben werden. Dieses wird durch den Befehl $51 beschleunigt. Alle Daten im Trackbuffer werden unverzüglich auf die Diskette geschrieben und abhängig vom Drive/Display Befehl ($44) wird der Motor nach erfolgtem Schreibvorgang sofort gestoppt.

KOMMANDO: $44

FUNKTION: Drive/Display Einstellung

AUX1: Konfigurationsbyte

AUX2: nicht benutzt

BESCHREIBUNG: Der Wert in AUX1 setzt das Drive/Display Byte im Laufwerk. Dieses Byte kann über keinen Befehl direkt ausgelesen werden, so daß immer alle Bits richtig gesetzt werden müssen.
Die einzelnen Bits beinhalten die folgenden Funktionen:
Bit 0 - BELL bei ERROR zulassen
Bit 1 - unbenutzt
Bit 2 - unbenutzt
Bit 3 - bei Kommando $51 Motor nicht ausschalten
Bit 4 - bei Kommando $20 VTOC+Boot Sektoren nicht schreiben
Bit 5 - Formatieren ohne Verify
Bit 6 - Trackanzeige in Hexadezimal
Bit 7 - ERROR-Anzeige zulassen

KOMMANDO: $4B

FUNKTION: Slow/Fast Konfiguration

AUX1: Konfigurationsbyte

AUX2: nicht benutzt

BESCHREIBUNG: Mit dem Wert in AUX1 wird das Drive-Slow-Status Byte des Laufwerkes beeinflußt. Dieses Byte kann über keinen Befehl direkt ausgelesen werden, so daß alle Bits richtig gesetzt werden müssen.
Die einzelnen Bits haben folgenden Funktionen:
Bit 0 - Read Sektor slow
Bit 1 - Write Sektor slow
Bit 2 - Kommando $57 Verify ausschalten
Bit 3 - Laufwerk vollständig im 'Slow-Mode'
Bit 4 - unbenutzt
Bit 5 - unbenutzt
Bit 6 - 1 Track slow (nach Trackwechsel 0)
Bit 7 - 1 Diskette slow (nach Diskettenwechsel 0)

KOMMANDO: $4C

FUNKTION: Direkter Sprungbefehl ohne Rückmeldung

AUX1: Sprungadresse LOW BYTE

AUX2: Sprungadresse HIGH BYTE

BESCHREIBUNG: Der Prozessor im Laufwerk wird durch diesen Befehl veranlaßt, direkt zur Speicherstelle zu springen, die sich in AUX1 und AUX2 befindet. Das Laufwerk gibt keine Rückmeldung an den Computer zurück, so daß eine Rückmeldung von dem Programm aus gegeben werden muß, zu dem der Prozessor gesprungen ist.

KOMMANDO: $4D

FUNKTION: Direkter Sprungbefehl mit Rückmeldung

AUX1: Sprungadresse LOW BYTE

AUX2: Sprungadresse HIGH BYTE

BESCHREIBUNG: Dieser Befehl gleicht dem Vorhergehenden bis auf den kleinen Unterschied, daß das Laufwerk vor ausführen des Programms eine Rückmeldung an den Computer gibt.

KOMMANDO: $41

FUNKTION: Kommandotabelle verlängern oder verkürzen

AUX1: nicht benutzt

AUX2: nicht benutzt

BESCHREIBUNG: Das Laufwerk erwartet 3 Bytes vom Computer. Das 1. Byte ist das Kommando. Das 2. und 3. Byte ist die Sprungadresse des über das neue Kommando erreichten Programmes in LOW/HIGH-Byte Format. Falls sich der neue Befehl schon in der Kommandotabelle befindet, wird dieser mit der neuen Startadresse versehen. Ist die Startadresse $0000 wird der Befehl aus der Kommandotabelle gelöscht.

KOMMANDO: $68

FUNKTION: Länge der SIO-Routine ermitteln

AUX1: nicht benutzt

AUX2: nicht benutzt

BESCHREIBUNG: Mit diesem Befehl wird die Länge der SIO-Routine ermittelt, die mit dem Befehl $69 aus dem Laufwerk in den Computer geladen wird. Das Laufwerk sendet 2 Bytes, die die Länge (LOW/HIGH) beinhalten.

KOMMANDO: $69

FUNKTION: SIO-Routine zum Computer senden

AUX1: Relokator-Adresse LOW BYTE

AUX2: Relokator-Adresse HIGH BYTE

BESCHREIBUNG: Dieser Befehl sendet die HIGH SPEED SIO-Routine zum Computer. Diese Routine wird bereits im Laufwerk zur Startadresse hin relokiert, die sich in AUX1 und AUX2 befindet.

KOMMANDO: $60

FUNKTION: Track schreiben

AUX1: Track Anfangssektor oder Anfangsadresse LOW BYTE

AUX2: Track Anfangssektor oder Anfangsadresse HIGH BYTE

BESCHREIBUNG: Die kompletten Daten für einen Track werden mit diesem Befehl auf die Diskette oder in den Trackbuffer geschrieben. Die Anzahl der zu übertragenden Bytes errechnet sich aus der Anzahl der Sektoren mal der Bytes pro Sektor. Wegen des sehr schwierigen Timings funktioniert dieser Befehl nur in normaler Übertragungsgeschwindigkeit.

KOMMANDO: $62

FUNKTION: Track lesen

AUX1: Track Anfangssektor LOW BYTE

AUX2: Track Anfangssektor HIGH BYTE

BESCHREIBUNG: Lesen eines kompletten Tracks mit einem Befehl von der Diskette oder aus dem Trackbuffer. Die Anzahl der zu erwartenden Bytes errechnet sich aus der Anzahl der Sektoren mal der Bytes pro Sektor.

```
0100 ;**************************************
0110 ;*  Demonstration 1 fuer Kommando $52  *
0120 ;**************************************
0130 ;
0140 DATBUF = $5000
0150 SECNUM = 1
0160 ;
0170     .OPT NO LIST
0180     .OPT OBJ
0190     *=  $4000
0200 ;
0210     LDA #$31    ; Bus ID
0220     STA $0300
0230     LDA #1      ; Laufwerks Nummer = 1
0240     STA $0301
0250     LDA #$52    ; Kommando $52
0260     STA $0302
0270     LDA #$40    ; Status fuer Daten lesen
0280     STA $0303
0290     LDA # <DATBUF ; Adresse fuer Datenbuffer Low
0300     STA $0304
0310     LDA # >DATBUF ; Adresse fuer Datenbuffer High
0320     STA $0305
0330     LDA #7      ; Wert fuer Timeout = 7 Sekunden
0340     STA $0306
0350     LDA #$80    ; 128 Bytes (in SD+MD) schreiben
0360     STA $0308
0370     LDA #0
0380     STA $0309
0390     LDA # <SECNUM ; Sector Nummer Low Byte
0400     STA $030A
0410     LDA # >SECNUM ; Sector Nummer High Byte
0420     STA $030B
0430     JSR $E459   ; Einsprung in die SIO-Routine im OS
0440     BMI ERROR
0450     CLS
0460     RTS
0470 ERROR SEC
0480     RTS
```

```
0100 ;**************************************
0110 ;*  Demonstration 2 fuer Kommando $52  *
0120 ;**************************************
0130 ;
0140 PRGBUF = $8000
0150 DATBUF = $5000
0160 ;
0170     .OPT NO LIST
0180     .OPT OBJ
0190     *=  $4000
0200 ;
0210     LDA #$31    ; Bus ID
0220     STA $0300
0230     LDA #1      ; Laufwerks Nummer 1
0240     STA $0301
0250     LDA #$52    ; Kommando $52
0260     STA $0302
0270     LDA #$40    ; Status fuer Daten lesen
0280     STA $0303
0290     LDA # <DATBUF ; Adresse fuer Datenbuffer Low
0300     STA $0304
0310     LDA # >DATBUF ; Adresse fuer Datenbuffer High
0320     STA $0305
0330     LDA #7      ; Wert fuer Timeout = 7 Sekunden
0340     STA $0306
0350     LDA #$80    ; 128 Bytes (in SD+MD) schreiben
0360     STA $0308
0370     LDA #0
0380     STA $0309
0390     LDA # <PRGBUF ; Adresse fuer Programmbuffer Low
0400     STA $030A
0410     LDA # >PRGBUF ; Adresse fuer Programmbuffer High
0420     STA $030B
0430     JSR $E459   ; Einsprung der SIO-Routine im OS
0440     BMI ERROR
0450     CLC
0460     RTS
0470 ERROR SEC
0480     RTS
```

```
0100 ;**************************************
0110 ;*  Demonstration 1 fuer Kommando $50  *
0120 ;**************************************
0130 ;
0140 DATBUF = $5000
0150 SECNUM = 1
0160 ;
0170     .OPT NO LIST
0180     .OPT OBJ
0190     *=  $4000
0200 ;
0210     LDA #$31    ; Bus ID
0220     STA $0300
0230     LDA #1      ; Laufwerks Nummer = 1
0240     STA $0301
0250     LDA #$50    ; Kommando $50
0260     STA $0302
0270     LDA #$80    ; Status fuer Daten schreiben
0280     STA $0303
0290     LDA # <DATBUF ; Adresse fuer Datenbuffer Low
0300     STA $0304
0310     LDA # >DATBUF ; Adresse fuer Datenbuffer High
0320     STA $0305
0330     LDA #7      ; Wert fuer Timeout = 7 Sekunden
0340     STA $0306
0350     LDA #$80    ; 128 Bytes (in SD+MD) schreiben
0360     STA $0308
0370     LDA #0
0380     STA $0309
0390     LDA # <SECNUM ; Sector Nummer Low Byte
0400     STA $030A
0410     LDA # >SECNUM ; Sector Nummer High Byte
0420     STA $030B
0430     JSR $E459   ; Einsprung der SIO-Routine im OS
0440     BMI ERROR
0450     CLC
0460     RTS
0470 ERROR SEC
0480     RTS
```

```
0100 ;**************************************
0110 ;*  Demonstration 2 fuer Kommando $50  *
0120 ;**************************************
0130 ;
0140 PRGBUF = $8000
0150 ;
0160     .OPT NO LIST
0170     .OPT OBJ
0180     *=  $4000
0190 ;
0200     LDA #$31    ; Bus ID
0210     STA $0300
0220     LDA #1      ; Laufwerks Nummer = 1
0230     STA $0301
0240     LDA #$50    ; Kommando $50 Sector ohne Verify schreiben
0250     STA $0302
0260     LDA #$80    ; Status fuer Daten schreiben
0270     STA $0303
0280     LDA # <DATBUF ; Adresse fuer Datenbuffer Low
0290     STA $0304
0300     LDA # >DATBUF ; Adresse fuer Datenbuffer High
0310     STA $0305
0320     LDA #7      ; Wert fuer Timeout = 7 Sekunden
0330     STA $0306
0340     LDA #$80    ; 128 Bytes (in SD+MD) schreiben
0350     STA $0308
0360     LDA #0
0370     STA $0309
0380     LDA # <PRGBUF ; Adresse fuer Programmbuffer Low
0390     STA $030A
0400     LDA # >PRGBUF ; Adresse fuer Programmbuffer High
0410     STA $030B
0420     JSR $E459   ; Einsprung der SIO-Routine im OS
0430     BMI ERROR
0440     CLC
0450     RTS
0460 ERROR SEC
0470     RTS
```

```
0480 ;
0490 DATBUF
0500     LDA #$FA
0510     JMP $FFA5   ; HEXOUT
```

```
0100 ;*************************************************
0110 ;*        Demonstration fuer Kommando $3F        *
0120 ;*  Uebertragungsrate fuer High-Speed ermitteln  *
0130 ;*************************************************
0140 ;
0150 DATBUF = $5000
0160 ;
0170     .OPT NO LIST
0180     .OPT OBJ
0190     *=  $4000
0200 ;
0210     LDA #$31    ; Bus ID
0220     STA $0300
0230     LDA #1      ; Laufwerks Nummer = 1
0240     STA $0301
0250     LDA #$3F    ; Kommando $3F
0260     STA $0302
0270     LDA #$40    ; Status fuer Daten lesen
0280     STA $0303
0290     LDA # <DATBUF ; Adresse fuer Datenbuffer Low
0300     STA $0304
0310     LDA # >DATBUF ; Adresse fuer Datenbuffer High
0320     STA $0305
0330     LDA #7      ; Wert fuer Timeout = 7 Sekunden
0340     STA $0306
0350     LDA #1      ; 1 Byte lesen
0360     STA $0308
0370     LDA #0
0380     STA $0309
0390     JSR $E459   ; Einsprung der SIO-Routine im OS
0400     BMI ERROR
0410     CLC
0420     RTS
0430 ERROR SEC
0440     RTS
```

```
0100 ;************************************
0110 ;*  Demonstration fuer Kommando $44  *
0120 ;************************************
0130 ;
0140     .OPT NO LIST
0150     .OPT OBJ
0160     *=  $4000
0170 ;
0180     LDA #$31    ; Bus ID
0190     STA $0300
0200     LDA #1      ; Laufwerks Nummer = 1
0210     STA $0301
0220     LDA #$44    ; Kommando $44
0230     STA $0302
0240     LDA #0      ; Status fuer keine Daten senden oder empfangen
0250     STA $0303
0260     LDA #7      ; Wert fuer Timeout = 7 Sekunden
0270     STA $0306
0280     LDA #$20    ; Bit 5 fuer Format ohne Verify
0290     STA $030A
0300     JSR $E459   ; Einsprung der SIO-Routine im OS
0310     BMI ERROR
0320     CLC
0330     RTS
0340 ERROR SEC
0350     RTS
```

```
0100 ;************************************
0110 ;*  Demonstration fuer Kommando $4B  *
0120 ;************************************
0130 ;
0140     .OPT NO LIST
0150     .OPT OBJ
0160     *=  $4000
0170 ;
0180     LDA #$31    ; Bus ID
0190     STA $0300
0200     LDA #1      ; Laufwerks Nummer = 1
```

```
0210     STA $0301
0220     LDA #$4B    ; Kommando $4B
0230     STA $0302
0240     LDA #0      ; Status fuer keine Daten senden oder empfangen
0250     STA $0303
0260     LDA #7      ; Wert fuer Timeout = 7 Sekunden
0270     STA $0306
0280     LDA #3      ; Bit 0+1 fuer Sector Read + Write Slow
0290     STA $030A
0300     JSR $E459   ; Einsprung der SIO-Routine im OS
0310     BMI ERROR
0320     CLC
0330     RTS
0340 ERROR SEC
0350     RTS
```

```
0100 ;************************************
0110 ;*  Demonstration fuer Kommando $4C  *
0120 ;* Einsprungbefehl. "C" - Complete   *
0130 ;* muss selbst gesendet werden !     *
0140 ;************************************
0150 ;
0160 GOADR = $FF5A   ; Einsprungadresse fuer "C" - Complete senden
0170 ;
0180     .OPT NO LIST
0190     .OPT OBJ
0200     *=  $4000
0210 ;
0220     LDA #$31    ; Bus ID
0230     STA $0300
0240     LDA #1      ; Laufwerks Nummer = 1
0250     STA $0301
0260     LDA #$4C    ; Kommando $4C
0270     STA $0302
0280     LDA #0      ; Status fuer keine Daten senden oder empfangen
0290     STA $0303
0300     LDA #7      ; Wert fuer Timeout = 7 Sekunden
0310     STA $0306
0320     LDA # <GOADR ; Einsprungadresse Low Byte
0330     STA $030A
0340     LDA # >GOADR ; Einsprungadresse High Byte
0350     STA $030B
0360     JSR $E459   ; Einsprung der SIO-Routine im OS
0370     BMI ERROR
0380     CLC
0390     RTS
0400 ERROR SEC
0410     RTS
```

```
0100 ;***************************************
0110 ;*   Demonstration fuer Kommando $4D    *
0120 ;* Einsprungbefehl. "C" - Complete wird *
0130 ;* vom Laufwerk sofort zurueckgesendet  *
0140 ;***************************************
0150 ;
0160 GOADR = $FF03   ; Einsprungadresse fuer "Drive Reset"
0170 ;
0180     .OPT NO LIST
0190     .OPT OBJ
0200     *=  $4000
0210 ;
0220     LDA #$31    ; Bus ID
0230     STA $0300
0240     LDA #1      ; Laufwerks Nummer = 1
0250     STA $0301
0260     LDA #$4D    ; Kommando $4D
0270     STA $0302
0280     LDA #0      ; Status fuer keine Daten senden oder empfangen
0290     STA $0303
0300     LDA #7      ; Wert fuer Timeout = 7 Sekunden
0310     STA $0306
0320     LDA # <GOADR ; Einsprungadresse Low Byte
0330     STA $030A
0340     LDA # >GOADR ; Einsprungadresse High Byte
0350     STA $030B
0360     JSR $E459   ; Einsprung der SIO-Routine im OS
0370     BMI ERROR
0380     CLC
0390     RTS
```

```
0400 ERROR SEC
0410     RTS

0100 ;************************************
0110 ;*  Demonstration 1 fuer Kommando $41  *
0120 ;*      Kommando $54 installieren      *
0130 ;************************************
0140 ;
0150     .OPT NO LIST
0160     .OPT OBJ
0170     *=  $4000
0180 ;
0190     LDA #$31    ; Bus ID
0200     STA $0300
0210     LDA #1      ; Laufwerks Nummer = 1
0220     STA $0301
0230     LDA #$41    ; Kommando $41
0240     STA $0302
0250     LDA #$80    ; Status fuer Daten schreiben
0260     STA $0303
0270     LDA # <COMBUF ; Adresse fuer Kommandobuffer Low
0280     STA $0304
0290     LDA # >COMBUF ; Adresse fuer Kommandobuffer High
0300     STA $0305
0310     LDA #7      ; Wert fuer Timeout = 7 Sekunden
0320     STA $0306
0330     LDA #3      ; 3 Bytes schreiben
0340     STA $0308
0350     LDA #0
0360     STA $0309
0370     JSR $E459   ; Einsprung der SIO-Routine im OS
0380     BMI ERROR
0390     CLC
0400     RTS
0410 ERROR SEC
0420     RTS
0430 COMBUF .BYTE $54 ; Kommando $54
0440     .WORD $8000 ; Einsprung $8000

0100 ;************************************
0110 ;*  Demonstration 2 fuer Kommando $41  *
0120 ;*        Kommando $3F loeschen        *
0130 ;************************************
0140 ;
0150     .OPT NO LIST
0160     .OPT OBJ
0170     *=  $4000
0180 ;
0190     LDA #$31    ; Bus ID
0200     STA $0300
0210     LDA #1      ; Laufwerks Nummer = 1
0220     STA $0301
0230     LDA #$41    ; Kommando $41
0240     STA $0302
0250     LDA #$80    ; Status fuer Daten schreiben
0260     STA $0303
0270     LDA # <COMBUF ; Adresse fuer Kommandobuffer Low
0280     STA $0304
0290     LDA # >COMBUF ; Adresse fuer Kommandobuffer High
0300     STA $0305
0310     LDA #7      ; Wert fuer Timeout = 7 Sekunden
0320     STA $0306
0330     LDA #3      ; 3 Bytes schreiben
0340     STA $0308
0350     LDA #0
0360     STA $0309
0370     JSR $E459   ; Einsprung der SIO-Routine im OS
0380     BMI ERROR
0390     CLC
0400     RTS
0410 ERROR SEC
0420     RTS
0430 COMBUF .BYTE $3F ; Kommando $3F
0440     .WORD $00   ; Kennung fuer Kommando loeschen

0100 ;*********************************
0110 ;* Demonstration fuer Kommando $60 *
0120 ;*     'Write Track' - Befehl      *
0130 ;*********************************
```

```
0140 ;
0150 TRKDAT = $5000  ; Adresse der kompletten Trackdaten
0160 TRKLEN = $0900  ; $900 SD, $D00 MD, $1200 DD
0170 SECTOR = 1      ; Anfangssektor eines Tracks
0180 ;
0190     .OPT NO LIST
0200     .OPT OBJ
0210     *=  $4000
0220 ;
0230     LDA #$31    ; Bus ID
0240     STA $0300
0250     LDA #1      ; Laufwerks Nummer = 1
0260     STA $0301
0270     LDA #$60    ; Kommando $60
0280     STA $0302
0290     LDA #$80    ; Status fuer Daten schreiben
0300     STA $0303
0310     LDA # <TRKDAT ; Trackdaten Low Byte
0320     STA $0304
0330     LDA # >TRKDAT ; Trackdaten High Byte
0340     STA $0305
0350     LDA #7      ; Wert fuer Timeout = 7 Sekunden
0360     STA $0306
0370     LDA # <TRKLEN ; Tracklaenge Low Byte
0380     STA $0308
0390     LDA # >TRKLEN ; Tracklaenge High Byte
0400     STA $0309
0410     LDA # <SECTOR
0420     STA $030A
0430     LDA # >SECTOR
0440     STA $030B
0450     JSR $E459   ; Einsprung der SIO-Routine im OS
0460     BMI ERROR
0470     CLC
0480     RTS
0490 ERROR SEC
0500     RTS
```

```
0100 ;***********************************
0110 ;* Demonstration fuer Kommando $62 *
0120 ;*     'Read Track' - Befehl       *
0130 ;***********************************
0140 ;
0150 TRKDAT = $5000  ; Adresse der kompletten Trackdaten
0160 TRKLEN = $0900  ; $900 SD, $D00 MD, $1200 DD
0170 SECTOR = 1      ; Anfangssektor eines Tracks
0180 ;
0190     .OPT NO LIST
0200     .OPT OBJ
0210     *=  $4000
0220 ;
0230     LDA #$31    ; Bus ID
0240     STA $0300
0250     LDA #1      ; Laufwerks Nummer = 1
0260     STA $0301
0270     LDA #$62    ; Kommando $62
0280     STA $0302
0290     LDA #$40    ; Status fuer Daten lesen
0300     STA $0303
0310     LDA # <TRKDAT ; Trackdaten Low Byte
0320     STA $0304
0330     LDA # >TRKDAT ; Trackdaten High Byte
0340     STA $0305
0350     LDA #7      ; Wert fuer Timeout = 7 Sekunden
0360     STA $0306
0370     LDA # <TRKLEN ; Tracklaenge Low Byte
0380     STA $0308
0390     LDA # >TRKLEN ; Tracklaenge High Byte
0400     STA $0309
0410     LDA # <SECTOR
0420     STA $030A
0430     LDA # >SECTOR
0440     STA $030B
0450     JSR $E459   ; Einsprung der SIO-Routine im OS
0460     BMI ERROR
0470     CLC
0480     RTS
0490 ERROR SEC
0500     RTS
```

```
0100 ;****************************************
0110 ;*  Lesen der SIO-Routine vom Laufwerk  *
0120 ;****************************************
0130 ;
0140 ADR =   $5000   ; Adresse fuer die SIO-Routine
0150 ;
0160     .OPT NO LIST
0170     .OPT OBJ
0180     *=  $4000
0190 ;
0200     LDA #$31    ; Bus ID
0210     STA $0300
0220     LDA #1      ; Laufwerks Nummer = 1
0230     STA $0301
0240     LDA #$68    ; Kommando $68
0250     STA $0302
0260     LDA #$40    ; Status fuer Daten lesen
0270     STA $0303
0280     LDA #8
0290     STA $0304   ; Adresse fuer Laengenbyte Low
0300     STA $0306   ; Wert fuer Timeout = 8 Sekunden
0310     LDA #3
0320     STA $0305   ; Adresse fuer Laengenbyte High
0330     LDA #2
0340     STA $0308   ; 2 Bytes lesen
0350     LDA #0
0360     STA $0309
0370     JSR $E459   ; Einsprung der SIO-Routine im OS
0380     BMI ERROR
0390     INC $0302   ; Kommando $69
0400     LDA # <ADR
0410     STA $0304   ; Target Adresse der SIO-Routine Low
0420     STA $030A   ; Original Adresse der SIO-Routine Low
0430     LDA # >ADR
0440     STA $0305   ; Target Adresse der SIO-Routine High
0450     STA $030B   ; Original Adresse der SIO-Routine High
0460     LDA #$40
0470     STA $0303   ; Status fuer Daten lesen
0480     JSR $E459   ; Einsprung der SIO-Routine im OS
0490     BMI ERROR
0500     CLC
0510     RTS
0520 ERROR SEC
0530     RTS
```

Bezugsmöglichkeit

Die Speedy-Erweiterung und auch andere 8-Bit Atari Geräte können auch heute noch käuflich erworben werden.

- Atari Bit Byter User Club   Ein Computerclub, der die alten 8-Bitter von Atari noch unterstützt.
- Atari Bit Byter User Club - Floppyservice   Diese Adresse ist bei der Suche nach Speedy, Bibomon und Floppy 1050 möglicherweise sehr hilfreich.